TOWARD AUTOMATING WEB PROTOCOL CONFIGURATION FOR A PROGRAMMABLE LOGIC CONTROLLER EMULATOR

THESIS

Deanna R. Fink, Civilian

AFIT-ENG-T-14-J-4

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-T-14-J-4

TOWARD AUTOMATING WEB PROTOCOL CONFIGURATION FOR A PROGRAMMABLE LOGIC CONTROLLER EMULATOR

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Deanna R. Fink, B.S.C.S.

Civilian

June 2014

AFIT-ENG-T-14-J-4

TOWARD AUTOMATING WEB PROTOCOL CONFIGURATION FOR A PROGRAMMABLE LOGIC CONTROLLER EMULATOR

Deanna R. Fink, B.S.C.S.
Civilian

Approved:

/signed/	29 May 2014
Barry E. Mullins, PhD (Chairman)	Date
/signed/	29 May 2014
Maj Jonathan W. Butts, PhD (Member)	Date
/signed/	29 May 2014
Juan Lopez Jr. (Member)	Date

AFIT-ENG-T-14-J-4

Abstract

Industrial Control Systems (ICS) remain vulnerable through attack vectors that exist within programmable logic controllers (PLC). PLC emulators used as honeypots can provide insight into these vulnerabilities. Honeypots can sometimes deter attackers from real devices and log activity. A variety of PLC emulators exist, but require manual configuration to change their PLC profile. This limits their flexibility for deployment. An automated process for configuring PLC emulators can open the door for emulation of many types of PLCs.

This study investigates the feasibility of creating such a process. The research creates an automated process for configuring the web protocols of a Koyo DirectLogic PLC. The configuration process is a software program that collects information about the PLC and creates a behavior profile. A generic web server then references that profile in order to respond properly to requests. To measure the ability of the process, the resulting emulator is evaluated based on response accuracy and timing accuracy. In addition, the configuration time of the process itself is measured. For the accuracy measurements a workload of 1000 GET requests are sent to the index.html page of the PLC, and then to the emulator. These requests are sent at two rates: Slow and PLC Break. The emulator responses are then compared to those of the PLC baseline.

Results show that the process completes in 9.8 seconds, on average. The resulting emulator responds with 97.79% accuracy across all trials. It responds 1.3 times faster than the real PLC at the Slow response rate, and 1.4 times faster at the PLC Break rate. Results indicate that the automated process is able to create an emulator with an accuracy that is comparable to a manually configured emulator. This supports the hypothesis that creating an automated process for configuring a PLC emulator with a high level of accuracy is feasible.

I would like to dedicate this work to my family. They have equipped me with the ability to succeed both in academia and beyond.

Acknowledgments

I would like to thank Dr. Barry Mullins for his support and guidance through this process. I would also like to thank Capt Robert Jaromin for his expertise on the subject.

Deanna R. Fink

Table of Contents

Page

Page

List of Figures

Figure Page

List of Tables

List of Acronyms

Acronym	Definition
ACE	automatically-configured emulator
CPU	central processing unit
CUT	component under test
ECT	emulation configuration tool
HMI	human machine interface
HTTP	Hypertext Transfer Protocol
ICS	industrial control system
IED	intelligent electronic device
MAC	media access control
MCE	manually-configured emulator
MSS	maximum segment size
MTU	maximum transmission unit
OUI	organizationally unique identifier
PLC	programmable logic controller
RTU	remote terminal unit
SCADA	supervisory control and data acquisition
SUT	system under test
TTL	time to live
VM	virtual machine

TOWARD AUTOMATING WEB PROTOCOL CONFIGURATION FOR A PROGRAMMABLE LOGIC CONTROLLER EMULATOR

I. Introduction

1.1 Overview

Industrial control systems (ICSs) are a critical element of the nation's infrastructure. Ensuring that these systems are secure is important. industrial control systems (ICSs) remain vulnerable to cyber-attacks, particularly due to unprotected programmable logic controllers (PLCs). A PLC is a component of ICSs that controls many of the system's physical functions such as, "logic, sequencing timing, counting, and arithmetic in order to control machines and processes"[Bol00]. A cyber-attack on a PLC can result in a catastrophic failure of the system.

One way in which more can be learned about the vulnerabilities and exploits associated with PLCs is through honeypot technology. Honeypots are "fake" versions of a system intended to fool an attacker into thinking they are on an actual system. The honeypots are then used to capture the attacker's activity. Honeypots take two forms: high-interaction and low-interaction. High-interaction honeypots are a real device, but serve no purpose other than to distract attackers and log information [PrH08]. Low-interaction honeypots are emulators of the real system and have many benefits, particularly in the case of PLC honeypots [PrH08]. They can be cost effective and readily accessible to deploy than high-interaction honeypots. This research focuses on low-interaction honeypots.

Researchers have used various methods to implement PLC honeypots. For example, one project used two virtual machines and a real PLC. The first virtual machine logs the attack information, while the second is a low-level honeypot emulating a PLC. The honeypot attaches to a real PLC of the same type that the honeypot is emulating [Dig11]. Another project used Honeyd, a program that specializes in creating large networks of honeypots [Pot05]. It includes a PLC package that offers a minimally functional emulator [Hon12].

A more recent independent research project created an emulator of a single PLC [Jar13]. The project focused on creating an emulator that was as accurate as possible. The emulator that the project created was a manually-coded emulator of a Koyo DirectLogic PLC [Koy13] [Jar13]. Because the emulator was configured manually, the process was time consuming. Having the ability to automatically configure the emulator to act like an arbitrary PLC device would create a more flexible tool for studying the PLC vulnerabilities.

1.2 Goals and Hypothesis

This research theorizes that an emulator of a PLC can be created through an automated process. The goal of this research is to create an emulator of the web protocols of a PLC using an automated configuration process. The resulting automatically-configured emulator (ACE) should respond with a goal of 100% accuracy to common requests. Accuracy is based on how closely the ACE responses match those of the PLC. Potentially, with continued research, a threshold level that consistently deceives the attackers can be achieved. This experiment seeks to answer the question: Can the process of configuring a PLC emulator be automated?

Automating this process lays a foundation for further honeypot research. With further production, it may potentially allow researchers to quickly create emulators of multiple honeypots. This could make it easier to deploy entire networks of honeypots into real industrial networks, which could then be used to log activity in the system. Another potential use is in academic settings. In order to educate people on the PLCs vulnerabilities. Emulators can easily be created for entire classrooms. Automating the process of configuring PLC web protocols is one step towards having complete emulators of various types of PLC that are flexible and quickly created.

1.3 Scope and Limitations

The emulation target for this research are the HTTP protocols of a Koyo DirectLogic 405 PLC. The protocols are limited to HTTP, due to the open knowledge of their implementation. In addition the ACE is tested only against the Koyo DirectLogic PLC. This research chose to focus on this PLC due to its low cost, availability of documentation, presence of known exploits, and its previous use in other research [Jar13]. That project created a manually-configured emulator of the Koyo DirectLogic PLC. The emulator included the HAP, Modbus, and HTTP protocols of the PLC. To emulate the HTTP protocols, the previous research used a Python-based HTTP web server. Because of the availability of libraries and modules, the same Python web server was chosen for this research. Emulating the same PLC with similar emulation software allows for some comparison of an automatically-configured emulator to a manually-configured emulator. Further details of the previous project are included in Section 2.4.2.

1.4 Approach

To create the configuration process, a set of rules and settings are compiled. These settings include network interface options and web server response behavior. During the configuration process, the configuration tool changes the values of settings based on information it gathers in real time from the PLC. The result of the process is an ACE. The usefulness of the configuration process is based on the accuracy of the emulator and the time required to configure the emulator. Accuracy is measured by two metrics, using a real PLC as a baseline. These levels are response accuracy and response time.

1.5 Thesis Overview

Chapter 2 gives background information on industrial control systems, honeypots, and related research. Chapter 3 describes the tools created for this study and how they are used to create an automatically-configured emulator. The methodology used for this research is provided in Chapter 4. Next, Chapter 5 discusses the results of the experiments. Finally, Chapter 6 covers the research conclusions of this study and offers possible avenues for future research.

II. Literature Review

2.1 Introduction

This chapter introduces current research in industrial control systems (ICSs) and the application of honeypots to ICS. Section 2.2 gives an overview of ICS, supervisory control and data acquisition (SCADA) systems, and their associated vulnerabilities. Section 2.3 introduces honeypots and explores various approaches available to implement them. Section 2.4 outlines current research in ICS honeypots and identifies areas for improvement. Finally, Section 2.5 summarizes the chapter.

2.2 Industrial Control Systems

ICSs control much of the machinery in the nation's manufacturing and critical infrastructure. This includes electrical systems, water, gas, transportation, and many other industries [SJK13]. ICSs include multiple types of control systems. This research is focused particularly on SCADA systems. Multiple devices make up industrial control systems: the Control Server (called the master terminal unit in SCADA systems), the remote terminal unit (RTU), intelligent electronic devices (IEDs), the Data Historian, the Input/Output Server, the human machine interface (HMI) and programmable logic controllers (PLCs) [SJK13]. PLCs are most relevant to this research.

2.2.1 Supervisory Control and Data Acquisition Systems.

One distinct characteristic of SCADA systems is their geographical disbursement [SFS13]. Figure 2.1 illustrates the general structure for a SCADA system. In a SCADA system, operators are able to perform real-time monitoring and control of industrial automation systems and industrial processes from remote locations. Some of the capabilities a SCADA system provides are:

- Access of sensor measurements of industrial processes

- Detect and correct process errors
- Measure trends over time
- Control geographically dispersed processes with a small, less specialized staff

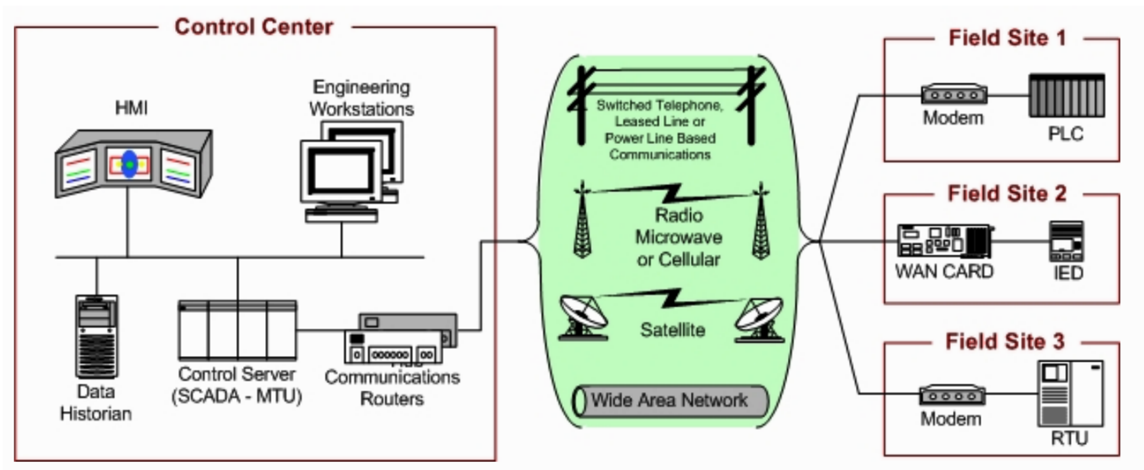


Figure 2.1: Structure of a Supervisory Control and Data Acquisition System [SJK13]

One element of a SCADA system is the HMI, which allows operators to connect to field devices and configure them. As shown in Figure 2.1 the HMI connects to field devices through telephone and power lines, radio, microwaves, cellular communications and satellite. In a SCADA system, these field devices can be an RTU, IED, or PLC. The PLC/IED/RTU contains logic that monitors sensors and instructs machinery on what actions to take. RTUs are field devices that often communicate information to the SCADA system through wireless communications [SJK13]. Occasionally the term RTU and PLC are used interchangeably, but this research considers them separate devices. An IED is a "'smart' sensor/actuator[s] containing the intelligence required to acquire data, communicate to other devices, and perform local processing and control" [SJK13]. PLCs are logic-based computers that perform a variety of controls. Further discussion of PLC functionality is described in Section 2.2.2.

Proper functioning of the PLC/IED/RTU requires low-latency communication between sensors, the Master Terminal Unit (MTU), and other PLCs [SJK13]. Any malfunction in these devices can result in downtime, evaluation, repair and possible destruction of the entire system. PLCs are vulnerable pieces of SCADA systems [Lev11]. This is a problem because PLCs control the major functions of an ICS.

Infrastructure and manufacturing began using industrial control systems before the era of Internet connectivity. First generation ICS architecture designs did not include interconnection with inter-networked computer systems. If remote connectivity was required, it was primarily through a non-persistent dial-up telephone connection. As the Internet grew in popularity, computer networking expanded using Local Area Networks (LAN) and Wide Area Networks (WAN). Industrial sectors began to recognize the potential benefits of using this type of networking for their control systems. The use of these systems improved efficiencies and reduced the costs for ICS. This resulted in the creation of SCADA systems [Lev11]. Because operators are now connecting to PLCs through a TCP/IP network connection, PLCs may be exposed to the Internet and susceptible to network-based intrusions.

2.2.2 Programmable Logic Controllers.

PLCs consist of four parts: an input/output device, central processing unit (CPU), a programming device, and a power supply [Roh96]. The purpose of a PLC is to use "programmable memory to store instructions and to implement functions such as logic, sequencing timing, counting, and arithmetic in order to control machines and processes" [Bol00]. PLCs have become more popular due to their many advantages. Some of these advantages include cost effectiveness, flexibility, computational abilities, troubleshooting aids, and component reliability [Jac08].

The PLC uses ladder logic to control and monitor systems. Operators can configure the ladder logic and other functions via the HMI. They can also connect the HMI to the PLC in various ways, including "a dedicated platform in the control center, a laptop on a wireless LAN, or a browser on any system connected to the Internet" [SJK13]. If an attacker is able to connect to a PLC through any of these avenues, he/she can read and potentially alter the logic program. Then the PLC would react incorrectly to sensor information, thus disrupting the process under control and potentially causing physical damage to equipment.

For example, in 2010 a computer worm called Stuxnet was discovered that targets Siemens PLCs. It had infiltrated an Iranian uranium plant by looking for Internet-facing computers using the Windows operating system that were also connected to ICS networks. Once the worm found the computers that fit the profile, it used them to gain access to PLCs. Attackers were able to reprogram PLCs on the Iranian ICS so that motors would spin at speeds not in line with their specifications. Moreover, the attackers installed a rootkit that prevented the true state of the motors from being reported to an operator [FMC11]. The combination of these two attacks caused physical damage that resulted in a great financial loss for the Iranian government.

Attacks like this on a nation's critical infrastructure could cause financial damage and potentially harm human life. One possible approach to learn how these types of cyber-attacks are executed, their associated exploitable vulnerabilities, and how to defend against them is through honeypots, which Section 2.3 describes.

2.2.3 Attack Vectors.

PLCs are a target for vulnerabilities in SCADA systems. Connectivity to TCP/IP networks increases their exposure [Cpn11]. As of 2007, the most common attack vector is through corporate networks and the Internet [BLK07]. SCADA systems are often connected directly to corporate WANs. Communication between SCADA devices and other computers within the corporate network may go unchecked. Because of this, a compromised computer sitting inside a corporate WAN bypasses all measures to protect SCADA systems, and becomes an open door to the PLCs.

Some SCADA systems are actually connected directly to the Internet, which opens them up to a wide array of attackers. This includes script kiddies (unskilled hackers) who may not intentionally direct sophisticated attacks at SCADA systems, but do have the ability to cause damage nonetheless [KLK09]. Additional attack vectors including dial-up modems, wireless systems, and virtual private networks add to the complexity of entry points, and make SCADA security a more difficult issue to address [BLK07]. The next section describes one potential method in which researchers can begin to create more efficient security solutions for SCADA systems.

2.3 Honeypots

2.3.1 Overview.

In general, a honeypot is a tool that can be used to log auditable information and often serves as a decoy that draws in attackers away from real target systems. Two basic types of honeypots exist, and each has their own merits. The first type, the high-interaction honeypot, is an actual duplicate device of the same type of system that the honeypot is meant to protect [PrH08]. The second type, which is discussed in greater detail and is the focus of this research effort, is a low-interaction honeypot. This type of honeypot consists of different hardware and software in order to emulate the device that the honeypot is protecting [PrH08].

2.3.2 High-Interaction Honeypots.

A high-interaction honeypot is useful in that it is able to function exactly as the device in question, because it employs the same hardware and software. An attacker interacting with a high-interaction honeypot should have no reason to question that it is any different from the actual targeted system.

Because high-interaction honeypots are configured exactly as real systems on the network, they are also vulnerable to the same attacks. Moreover, duplicate real devices are typically more expensive and difficult to maintain than other options. In addition, many companies are unwilling to share information about their systems [DHS13], because much of it is proprietary [ByL04]. The next section describes an alternative honeypot solution for SCADA research.

2.3.3 Low-Interaction Honeypots.

Low-interaction honeypots, on the other hand, are an emulation of the device. The term emulation generally means "using some device or program in place of a different one to achieve the same effect as using the original" [Sla03]. Emulation of software and applications can be difficult due to their sometimes proprietary nature [HoW05]. Therefore, emulators may only mimic functionality with either entirely new systems, or only pieces of the original system [Jul91]. Although emulators often provide only a subset of services, when used as honeypots, they only need to provide enough interaction that an attacker or tool accepts the emulator as the real device [PrH08].

The main purpose of a low-interaction honeypot is to serve as a facade of interaction, and in most cases attackers can do little to no harm to them. In addition, low-interaction honeypot solutions are often less expensive [PrH08], and more practical to operate and configure than the devices they are designed to represent. Also, because there is no real operating system, it is less likely that a low-interaction honeypot can be used to attack other systems within the network [Spi03].

Despite their many advantages, low-interaction honeypots are much more difficult to disguise. Because they are not the actual device, designers must incorporate features and customize them to behave as closely as possible to the devices they are intended to mimic. These steps include simulating network protocols, spoofing device specifications, and more in depth, simulating the way that it would interact with other devices within its network [PrH08]. The next section discusses ways of implementing some of these characteristics, and how each implementation could affect the ability to add additional features.

2.4 Honeypot Research

2.4.1 Current Honeypot Activities.

A variety of projects exist that demonstrate a range of efforts to utilize honeypot technology in industrial control systems. Interestingly, in some way, each of them is deficient and leaves room for improving honeypot technology. The first project, called the SCADA HoneyNet Project, attempted to accomplish many of the same goals that this research effort outlines. The developers planned to use Honeyd (discussed in more detail in Section 2.4.3) to emulate multiple PLCs. The developers were able to simulate a small set of functionality, but development of the SCADA HoneyNet project stopped in 2005 [PoF05].

Another project, created by Digital Bond in 2008, took a different path. The project incorporated many of the ideas in the SCADA HoneyNet project, and used two virtual machines. The first virtual machine is a Honeywall VM that logs traffic information, while the second is a low-level honeypot emulating a Modicon Quantum PLC [Dig11]. As shown in Figure 2.2, the Honeywall logs and filters traffic as it comes in from the Internet. The emulator virtual machine runs Honeyd to emulate the PLC services. Digital Bond also extended the project to allow a real PLC to be attached to the system. In this case, the PLC replaces the emulator VM [Dig11]. This project is one of the most developed so far, but requires many pieces of hardware in order to operate it.

In 2011, a student at Iowa State created a Honeynet in an attempt to log attack information on PLCs [Wad11]. The study utilized the Digital Bond Design. The Honeynet was deployed on a network for approximately one month and monitored attacks. The

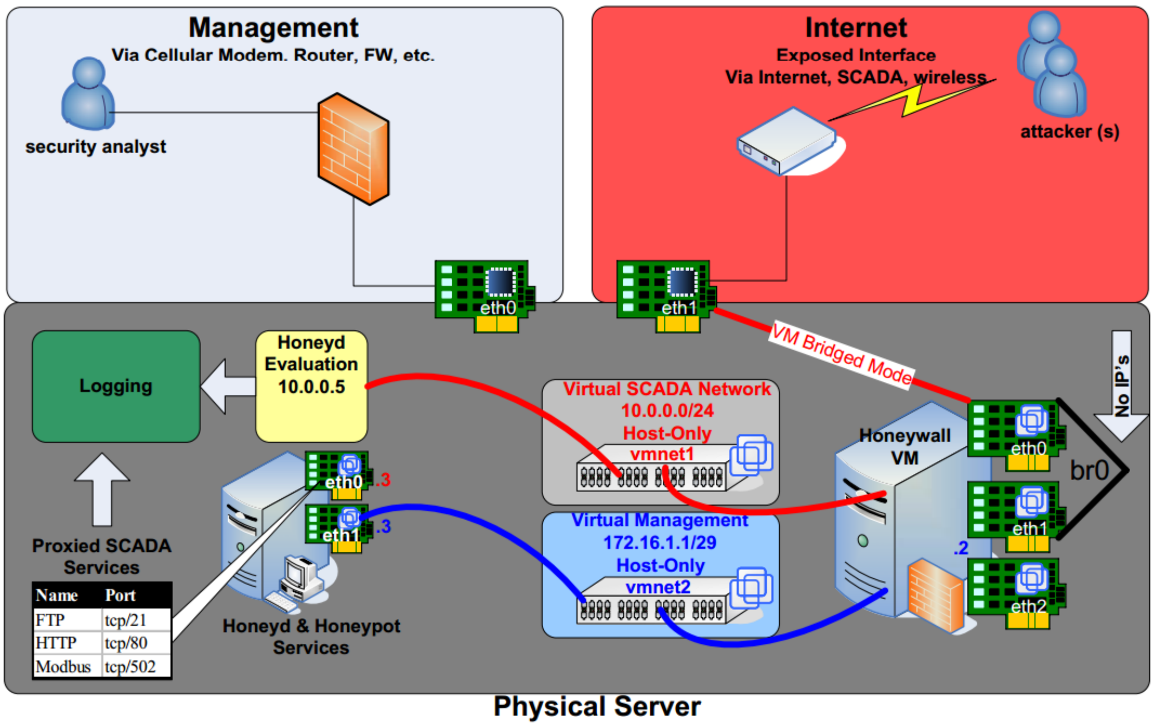


Figure 2.2: Structure of the Digital Bond Honeynet [Dig11]

study did not result in any logged attacks. However, two critical pieces of the experiment may mean that the network was not an accurate representation of a real PLC network. First, the Honeynet was on the 129.186.255.255 subnet, which is identifiable as an Iowa State domain from a Whois lookup. If an outsider were to see this, it would provide strong evidence that the honeypot was not within a real industrial network. Second, the emulated PLC was a Schneider Quantum Modicon which was the default configuration for the Digital Bond honeypot. If an attacker visiting the network knew about the Digital Bond experiment, seeing that the honeypot was a Modicon PLC might have deterred the attacker from exploiting the device.

Creating a PLC honeypot is difficult. Contributing to the difficulty is the large number of PLC brands, models, and protocols that exist which are often proprietary. No one has yet perfected a process, but research continues to combine and build upon techniques, making fully functioning PLC honeypots more feasible. The next section outlines another project that uses single board computers to implement a honeypot.

2.4.2 Independent Gumstix Research.

A researcher at the Air Force Institute of Technology developed a manually-configured emulator (MCE) to resemble the behavior of a Koyo DirectLOGIC 405 PLC on a single board computer called a Gumstix [Gum12]. The researcher modified the way in which the device (running Linux) accepts and responds to network queries [Jar13].

When a query arrives at the device, it must go through a series of filters so the device will be able to run the correct services and respond appropriately. Here, appropriately means that the device will respond in the same way that the Koyo PLC is expected to respond.

As shown in Figure 2.3, in the boxes labeled *rules*, the packets are filtered through IP Tables rules to determine how the packet should be handled. The filter rejects any TCP packet that arrives on a port other than port 80 or 502. Any incoming packets that appear to be from a port scanner are dropped and a custom response packet is formed. When generating a response, the firewall must insert information that the port scanner (e.g., Nmap) is expecting. The firewall makes a decision based on the technique Nmap uses to interrogate the device during the scanning process [Nma12]. In this case, the packet will contain information to make the Gumstix device look like a Koyo PLC. The response packet then moves through the kernel and out through the outgoing IP Tables rules [Jar13].

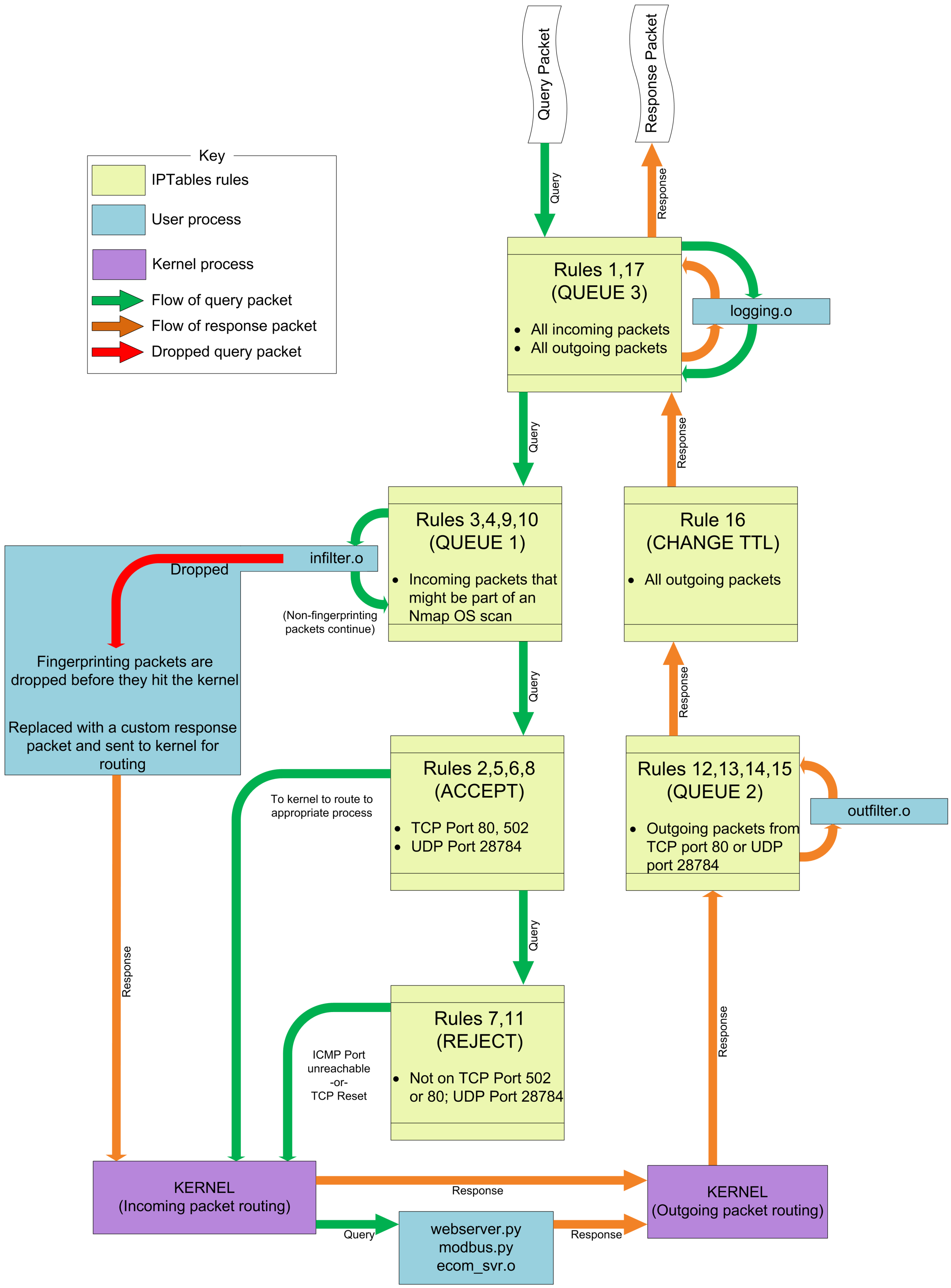


Figure 2.3: Gumstix Emulation Filter of Koyo DirectLOGIC 405 PLC [Jar13]

If the query is a TCP packet sent to port 80 or 502, or a UDP packet sent to port 28784, then the kernel routes the packet to its corresponding, user-space process [Jar13]. To move the packets from kernel-space to user-space, the emulator uses *NFQUEUE*. This is a kernel module that allows the packets to be sent to user-space processes through queues. The specifications for the Koyo PLC determine the routing for accepted packets [Jar13].

This process was demonstrated to work well, in that the Gumstix computer reacts to various queries in the same manner that the Koyo PLC would. The Gumstix emulator is currently limited to behave like a single PLC. It is not able to emulate other manufacturers or models of PLCs, nor is it able to emulate more than a single instance of a PLC. The former functionality is achievable, but currently this would require the developer to hardcode the configuration for all of the PLC's interactive query/response behavior. This research incorporates parts of the Gumstix MCE design into the creation of an ACE.

2.4.3 Honeyd.

In 2003, Niels Provos began a project called Honeyd. The project was an effort to create a framework for the development of low-interaction honeypots, with a focus on emulating multiple honeypots on a single computer [Hon12]. In contrast to the Gumstix architecture, Honeyd runs with root privileges, eliminating the transition from the kernel to user mode and back to the kernel [PrH08]. In addition, Honeyd has a feature that allows it to route traffic to a range of IP addresses, as specified by the developer [PrH08].

Figure 2.4 illustrates how network traffic flows directly to Honeyd. If the developer prefers to emulate multiple devices, then he can allocate a set of IP addresses to Honeyd [PrH08]. Honeyd will then route traffic to the appropriate honeypot. Additionally, the developer has the flexibility to configure each honeypot to emulate a variety of devices. When the packet arrives at the dispatcher, Honeyd performs a series of checks, similar to the Gumstix architecture, that allows Honeyd to respond to the query properly and perform the correct services. Honeyd refers to the specifications of how a honeypot should respond as the personality of the honeypot. After Honeyd executes the correct services, any packet that requires a response, will go through the routing process and then to the personality engine. The personality engine then modifies the response to fit the personality of the honeypot that received the query [PrH08].

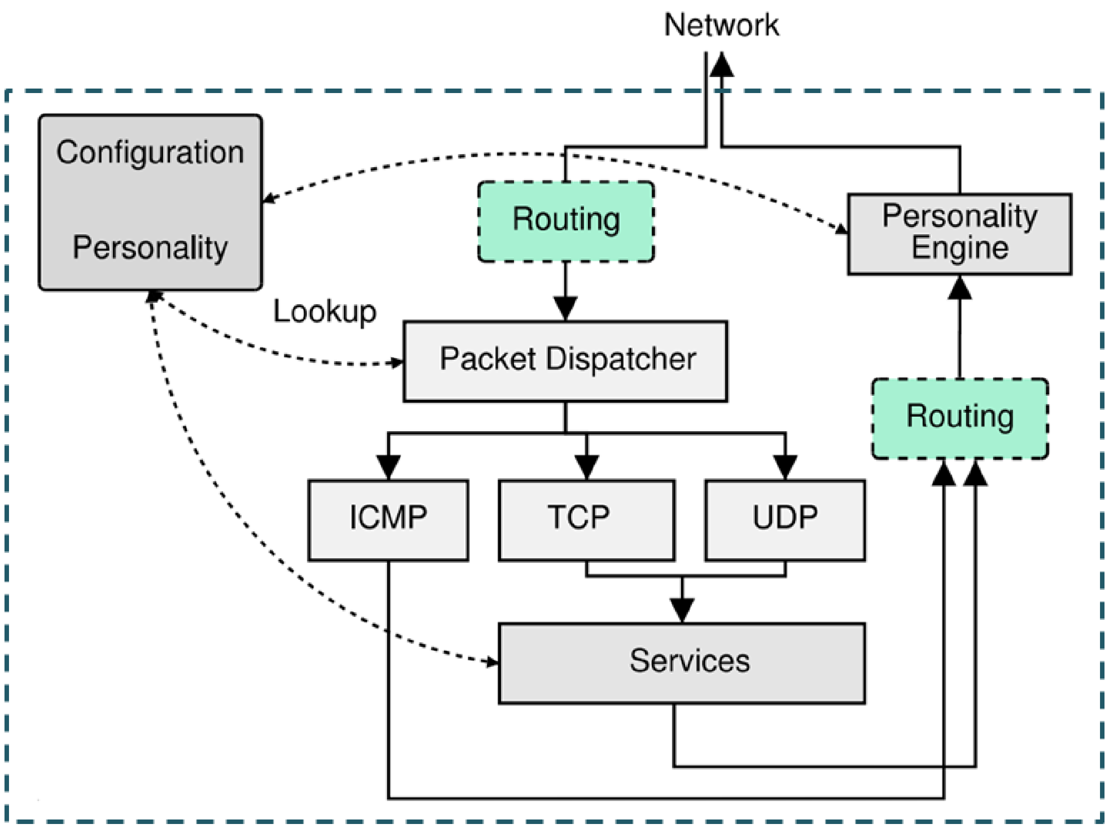


Figure 2.4: Honeyd Architecture [PrH08]

The range of functionality that Honeyd offers makes it an attractive option for emulating multiple PLCs, but it has drawbacks that the Gumstix architecture does not. First, Honeyd's popularity opens it to potential fingerprinting. Oberheide and Kair describe two methods to detect Honeyd [ObK06]. The first method is through default response messages. If these messages were left as the default, they provide an effective method to determine they are interacting with a honeypot. Second, there is an error in the way that Honeyd handles fragmented packets. When fragmented packets are sent to Honeyd, the protocol is considered incorrect and Honeyd will improperly respond to the packets with a SYN/ACK [ObK06]. Both of these issues can be corrected, but are considered added complexity when considering Honeyd for research.

2.5 Summary

This chapter discusses SCADA systems and why vulnerabilities exist in them. It, also, describes honeypots which could be used to learn more about how these vulnerabilities are being exploited. The chapter then provides an overview of various methods researchers used to develop honeypots for Industrial Control Systems.

III. Emulation Configuration Tool

This chapter describes the software that allows for the creation of a PLC web interface emulator. The emulation configuration tool (ECT) includes two pieces: the configuration process and the automatically-configured emulator (ACE). The configuration process works by using a set of configuration programs to gather information and create a profile. The emulator uses that information to run a web server. As shown in Figure 3.1 the PLC connects directly to the configuration machine through an Ethernet cable in order to communicate.

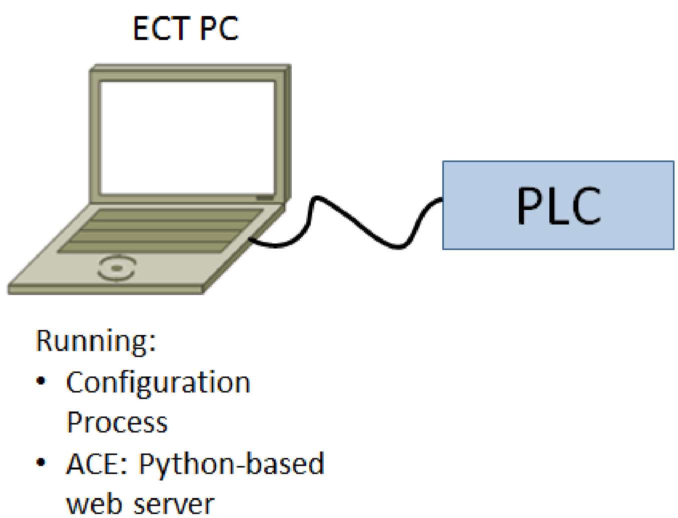


Figure 3.1: Configuration Setup

3.1 Design

The design of the ECT includes several considerations: speed, minimal user interaction, and accuracy. The purpose of the ECT is to automate the configuration process for PLC honeypots. Therefore, the ECT must be reasonably fast and require little human interaction, while maintaining an acceptable level of accuracy.

Based on the expected applications of this process, a maximum configuration time of five minutes is the threshold for this research. For instance, this tool could be used in an educational environment. Five minutes per emulator would be a reasonable time frame to set up a lab environment. Because of the speed and cost, each student could have an emulator to interact with. If the ECT is used for deploying emulators in an industrial environment, five minutes is also acceptable. It is expected that only a few PLCs would need to be created for use as honeypots within a single network. In addition the emulators would likely need to be created only once. A configuration time of five minutes would not keep operators from other work for an unacceptable amount of time.

In addition to speed, minimal user interaction is an important factor in the design of the ECT. The ECT requires few instances in which a human must interact and those user interactions would need little specialized skill. Whether this tool is used in an educational environment or being deployed in an industrial network, the individual setting up the emulator does not need to be an expert in PLC design and protocols. The commands needed to run the emulator should only require general technical knowledge. The ideal scenario for this research is a single script to run the entire process.

Finally, the ECT must be accurate. In order to create a profile for the responses of the PLC, the ECT collects information from three vectors: (1) web page collection, (2) mirroring of Hypertext Transfer Protocol (HTTP) response headers, and (3) configuration of network settings through pcap parsing. The combination of these methods increases the accuracy across multiple levels.

3.2 Configuration Process

To gather the necessary information, the configuration tool runs a variety of scripts which send and receive web traffic. The configuration process utilizes four creation files which probe the PLC and create the profile. These files are auto-wget.py, inputFinder.py, parse_pcap.py, and config_emulator.sh. A description of the configuration steps is shown in Figure 3.2. The steps of this process are outlined below. In parentheses, following each step, is the name of any scripts or files associated with that step.

1. Gather all of the web pages available on the interface. This step uses a third party tool called wget which harvests all of the pages and then saves them in a local directory. (*auto-wget.py*)

2. Send HTTP GET request to each page in the directory from step one. In addition, save response headers to a dictionary. (*inputFinder.py, variableFile.py*)

3. Record traffic from step two using tcpdump and save traffic capture to a *pcap* file.

4. Use the *pcap* file to gather network configuration information (*parse_pcap.py*)

5. Echo network configuration values to associated files within the Linux file system. (*config_emulator.sh*)

3.2.1 auto-wget.py.

This file runs the wget utility and the *inputFinder.py* program which is outlined in Figure 3.3. When *auto-wget.py* is executed, the operator must input the IP address of the PLC and any login credentials required to access pages in the web interface. It then runs the wget command **`wget -r -l 15 -e [optional credentials] robots=off [IP address]`**. The results of the wget command are a collection of all of the accessible pages in the web interface. Those pages are then placed in a folder. The label for this folder is the IP address of the PLC. In addition, auto-wget.py initializes the inputFinder.py program.

Before running *auto-wget.py* the user must start tcpdump using the command **`$sudo tcpdump -i eth0 -w settings.pcap`**. This records the TCP handshake packets that contain network settings required to configure the emulator VM. Once *auto-wget.py*

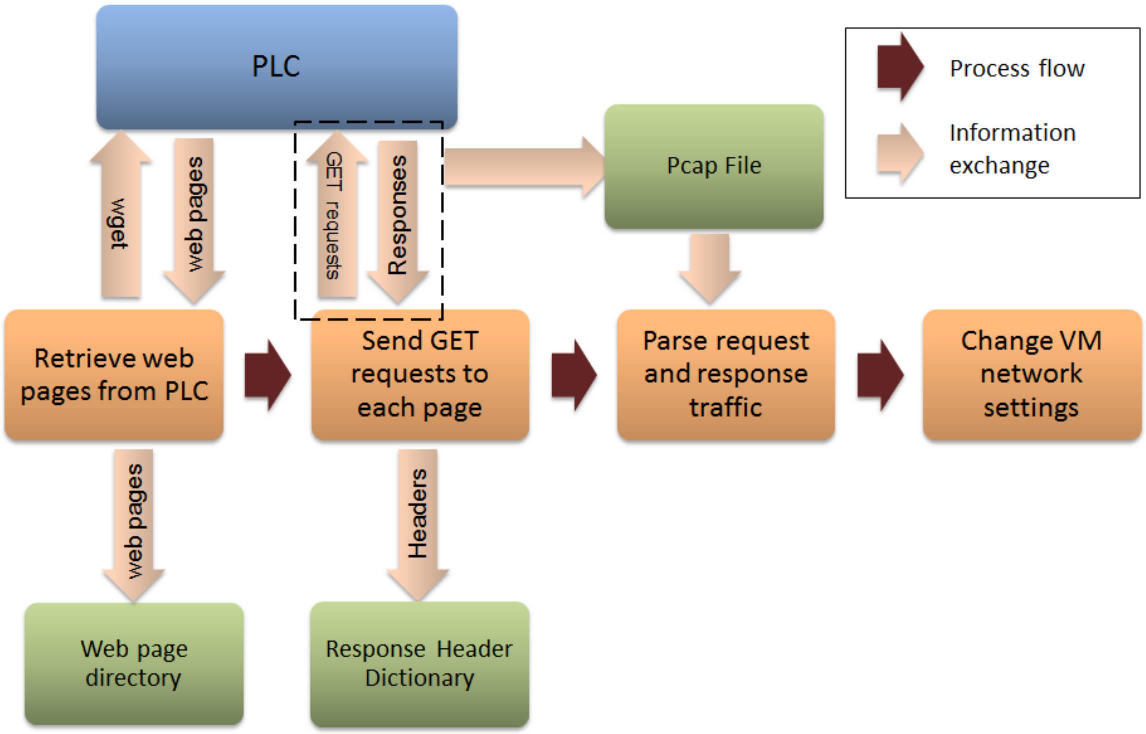


Figure 3.2: Configuration Process

has completed execution, the tcpdump script is ended by the user. Further discussion of how *settings.pcap* is used is discussed in Section 3.2.3.

3.2.2 inputFinder.py.

For the web server to properly handle requests for pages, it has to know how the PLC handles those requests. These requests include the headers of the response packets. The file that gathers this information is *inputFinder.py*. This program records responses from GET requests. As shown in Figure 3.4, a request is sent to the web page. The program iterates through each section of the response header. It builds a string from these sections. To send the requests it uses *httplib*, a built-in Python library. The HTTPConnect module allows the program to send requests to the PLC, record the responses, and retrieve header information.

```
import os, sys, subprocess, re, getpass, string, time


ip = raw_input("IP of PLC: ")    User inputs IP address
has_password = raw_input("Does the PLC require a password? (y/n): ")


if has_password == 'y' or has_password == 'Y':
    print "Please input the login credentials: "
    username = raw_input("Username: ")
    password = getpass.getpass()
    print "wget -r -l 15 -e --user="+username+" --password="+password+"
robots=off http://"+ip

    start = time.time()
    os.system("cd ~/Desktop/generic_emulator")
    os.system("wget -r -l 15 -e --user="+username+" --password="+password+"
robots=off http://"+ip)

else:
    start = time.time()
    os.system("cd ~/Desktop/generic_emulator")
    os.system("wget -r -l 15 -e robots=off http://"+ip)    Run wget command


varFile = open("variableFile.py", "w")    Open variableFile for writing


os.system("cd ~/Desktop/generic_emulator/lib")

os.system("python ./inputFinder.py "+ip)    Run inputFinder.py


end = time.time()
total_time = end-start
print "Total time: ", total_time
```

Figure 3.3: *auto-wget* Script

The headers from the responses are stored in a Python dictionary along with their associated web page. This dictionary is stored in the file *variableFile.py* which is described in Section 3.3.1. The web server portion of the emulator uses this dictionary to formulate its own responses. The key in the response dictionary is the file name of the web page. The value is the response header string that corresponds to the web page. Figure 3.5 shows a section of the response dictionary.

```
# os.walk will walk the entire directory of files and create a generator
object that can be iterated
for root, dirs, files in os.walk(htmlPath):

  for fileName in files:
    time.sleep(.001) #sleep if necessary

    filePath = str(root+'/'+fileName)
    preamble, realPath = root.split(ipaddress,1)

    h1.request("GET", realPath+'/'+fileName)

    r1 = h1.getresponse()

    allHeaders = r1.getheaders()
    headerString = ""
    for header in allHeaders:
        headerString = headerString+header[0]+': '+header[1]+'\r\n'

    toWrite = ("HTTP/1.0" if r1.version == 10 else "HTTP/1.1")+" "
    toWrite += str(r1.status)+" "
    toWrite += str(r1.reason)+'\r\n'
    toWrite += headerString+'\r\n'
    if ipaddress in toWrite:
        toWrite = toWrite.replace(ipaddress,"127.0.0.1")

    responses[realPath+'/'+fileName] = toWrite
```

Send a GET request to each page and retrieve the response

Compile header string

Place header in the response dictionary

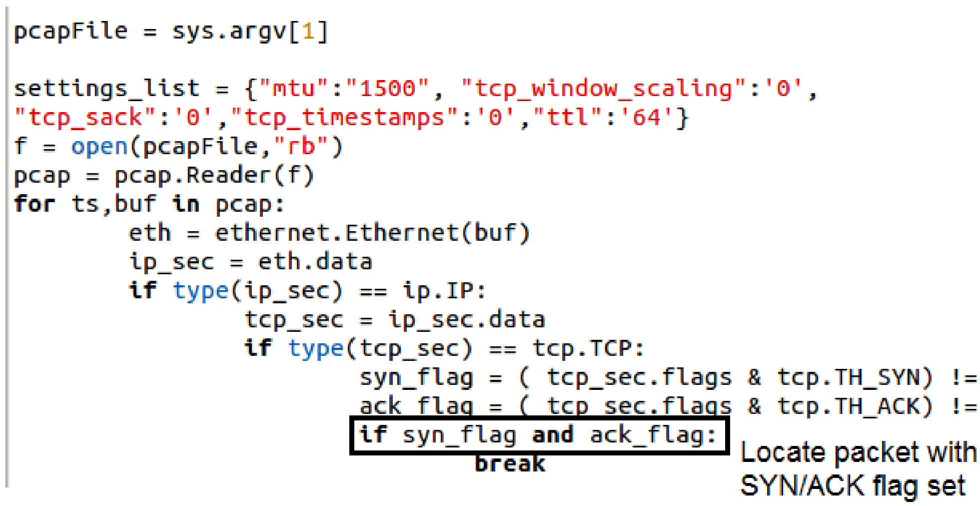
Figure 3.4: Sending GET Requests to Each Page and Recording Responses in *inputFinder*

```
responses = {'/setip.html': 'HTTP/1.1 200 OK\r\ncontent-type: text/html\r\n\r
\n', '/advanced.html': 'HTTP/1.1 200 OK\r\ncontent-type: text/html\r\n\r\n', '/
getp2p5.html': 'HTTP/1.1 200 OK\r\ncontent-type: text/html\r\n\r\n', '/
getp2p3.html': 'HTTP/1.1 200 OK\r\ncontent-type: text/html\r\n\r\n', '/
index.html': 'HTTP/1.1 200 OK\r\ncontent-type: text/html\r\n\r\n', '/
```

Figure 3.5: Response Dictionary in *variableFile.py*

3.2.3 parse_pcap.py and config_emulator.sh.

The *parse_pcap.py* script, combined with the config_emulator.py program, accomplishes the next step of configuration. This script collects information from the *settings.pcap* file created during the interaction between the PLC and the *auto-wget.py* script. Figure 3.6 shows that an initial dictionary is created containing the network settings to be changed. This includes the maximum transmission unit (MTU), TCP Window Scaling, TCP Selective Acknowledgment, TCP Timestamp, and time to live (TTL).



```python
pcapFile = sys.argv[1]

settings_list = {"mtu":"1500", "tcp_window_scaling":'0',
"tcp_sack":'0',"tcp_timestamps":'0',"ttl":'64'}
f = open(pcapFile,"rb")
pcap = pcap.Reader(f)
for ts,buf in pcap:
        eth = ethernet.Ethernet(buf)
        ip_sec = eth.data
        if type(ip_sec) == ip.IP:
                tcp_sec = ip_sec.data
                if type(tcp_sec) == tcp.TCP:
                        syn_flag = ( tcp_sec.flags & tcp.TH_SYN) != 0
                        ack_flag = ( tcp_sec.flags & tcp.TH_ACK) != 0
                        if syn_flag and ack_flag:
                                break
```


Figure 3.6: Locating SYN/ACK Packet with *parse_pcap.py*

The *parse_pcap.py* uses a Python library called *dpkt* to parse the pcap file. It specifically looks for a packet from the PLC with both the SYN and ACK flags set. This packet contains information about the network configuration of the PLC.

As Figure 3.7 shows, *parse_pcap.py* uses *dpkt*, again, to search for the options listed above and the values associated with them. The *parse_pcap.py* program then runs the *config_emulator.sh* bash script, with the network options as arguments to the script.

The *config_emulator.sh* script echoes the values to the appropriate files. Figure 3.8 shows that the current options being changed are in the ipv4 folder within Linux. Note that the ICMP Rate Limit is also a value being changed. The *dpkt* library does not have the functionality to retrieve this value. For this study, the rate limit is turned off. Pilot studies show that this value is turned off in other PLCs, so this is considered acceptable. A

```
settings_list["ttl"] = str(ip_sec.ttl)

tcp_options = tcp.parse_opts(tcp_sec.opts)
for option in tcp_options:
        if option[0] == 2:
                mtu = str(int(struct.unpack(">H",option[1])[0])+40)
                settings_list["mtu"] = mtu
        if option[0] == 3:
                wscale = str(struct.unpack(">H",option[1])[0])
                settings_list["tcp_window_scaling"] = wscale
        if option[0] == 4:
                sackok = str(struct.unpack(">H",option[1])[0])
                settings_list["tcp_sack"] = sackok
        if option[0] == 8:
                timestamp = str(struct.unpack(">H",option[1])[0])
                settings_list["tcp_timestamps"] = timestamp


os.system("./config_emulator.sh "+settings_list["mtu"]+" "+settings_list
["tcp_sack"]+" "+settings_list["tcp_window_scaling"]+" "+settings_list
["tcp_timestamps"]+" "+settings_list["ttl"])
```

Figure 3.7: Locating Network Options in *parse_pcap.py*

full list of the values changed by *config_emulator.sh* and their descriptions is included in Appendix C.

```
sudo ifconfig eth0 down
sudo ifconfig eth0 mtu $1
sudo ifconfig eth0 up
sudo echo "0">> /proc/sys/net/ipv4/icmp_ratelimit
sudo echo $2>> /proc/sys/net/ipv4/tcp_sack
sudo echo $3>> /proc/sys/net/ipv4/tcp_window_scaling
sudo echo $4>> /proc/sys/net/ipv4/tcp_timestamps
sudo echo $5>> /proc/sys/net/ipv4/ip_default_ttl
```

Figure 3.8: *config_emulator* Script

3.3 Emulator

Once the creation programs have completed, the emulator is ready to be started. The emulator is made up of a python web server, the web page files, and a file that contains dictionaries for the web server to refer to when formulating a response. The web sever is contained in the file *webserver.py*. The web page files are those gathered in the *inputFinder* program. Finally, the file that holds the response dictionary is *variableFile.py*.

3.3.1 webserver.py and variableFile.py.

The emulator uses a Python-based HTTP server to serve responses to the client [Pyt14]. This server is contained in *webserver.py*. The web server accepts GET requests from a client and serves responses based on the values within *variableFile.py*. The *variableFile.py* file contains a response dictionary that holds a mapping of the page names to the response given by the PLC when a GET request is sent to that page. Figure 3.9 shows the section of the web server code that handles GET requests. The GET request handler uses the file name as the key to search for the appropriate header in the response dictionary in *variableFile.py*. The header value that is returned is combined with the contents of the web page file. The resulting string is placed in a packet and sent as the response.

In pilot studies, a difference in response times between the PLC and the ACE was discovered. The average delay was 1.5 ms. Therefore, a static artificial delay of 1.5 ms is added to the request handler of the ACE web server for each response. This added delay is not as accurate as it could be. A more accurate delay would potentially be variable, to match the variability in the PLC responses. In addition the delay could also be based on the times of the responses gathered during the configuration process. This would allow for a delay that is tailored to the network speeds and the specific PLC. The response accuracy of the ACE with the delay is analyzed in addition to the response accuracy without the delay.

```
f = open(curdir + sep + self.path)

fileText = f.read()

self.wfile.write(responses[req.path]+fileText)

self.close_connection = 1
f.close()

return
```

Concatenate response header to web page text. Respond to GET request with the resulting string

Figure 3.9: Method for Responding to GET Requests Within *webserver.py*

3.4 Summary

The ECT combines speed, minimal interaction, extensibility and accuracy to create a useful configuration tool. It uses information gathered from *wget*, behavior of responses to GET requests, and system configuration options gathered from packet headers. This process creates a profile that fits the behavior of the PLC. In addition, more rules could be added if analysis shows that those rules would increase the capabilities of the ECT or the accuracy of the ACE.

IV. Methodology

4.1 Problem Definition

4.1.1 Goals and Hypothesis.

The goal of this research is to create a process for automatically configuring a PLC emulator. Some PLC emulators in existence are manually configured, which is a lengthy and specialized process. In addition, the resulting emulator is configured to a single type of PLC. An automated process would decrease the configuration time and allow for the configuration of an emulator that is able to mimic any type of PLC. In addition, although the process would be consistent over multiple iterations, the lack of human involvement can lead to a possible loss of accuracy.

It is hypothesized that an PLC emulator can be created through an automated process. Because of the large number of PLC types and the variety of protocols and implementations PLCs use, this research is focused on automating the process for configuring the emulator at the web-protocol level. This includes emulating open protocols used to control PLCs through a web interface. How well the process works is determined by the accuracy of the emulator and the efficiency of the process. Questions that this research addresses include:

- Can the process of configuring a PLC emulator be automated?
- Can the process be a step towards creating emulators using various types of PLC devices?

4.1.2 Approach.

To create a PLC emulator automatically, the configuration tool creates a profile by interacting with the PLC and collecting information about its behavior. This behavior includes the way in which the PLC responds to HTTP GET requests, and the network configuration of the PLC. The details of this process are outlined in Chapter 3.

Because many of the protocols associated with PLCs are proprietary, this research focuses only on the open TCP/IP protocols. Specifically the research emulates a web server that accepts traffic on port 80. In order to create a more complete emulation of a PLC, all protocols would need to be emulated. Emulating the TCP/IP protocols is a first step toward a full emulator.

4.2 System Boundaries

The system under test (SUT), shown in Figure 4.1, is the ECT. The components of the system are the configuration process and the resulting automatically-configured emulator (ACE). The configuration process is the component under test (CUT). The workload into the system are GET requests sent to the ACE. The ACE uses the response accuracy and response time metrics to evaluate its performance. The configuration process is evaluated based on configuration time.

4.3 System Services

The service ECT provides is the creation of a PLC emulator. There are three possible outcomes for this service: creation of an accurate emulator, creation of an inaccurate emulator, and no creation of an emulator. When the tool succeeds, an emulator is created that meets the threshold for acceptable accuracy. The measurements for accuracy are described in Section 4.5. If an emulator is created, but does not meet those standards, it is considered a failure. This outcome can occur either because outside factors affected the ability of the configuration process to complete successfully, or because the algorithm for creating the emulator is flawed. If no emulator is created, this is also a failure. This could occur if the ECT does not include enough information to create a functional emulator. It

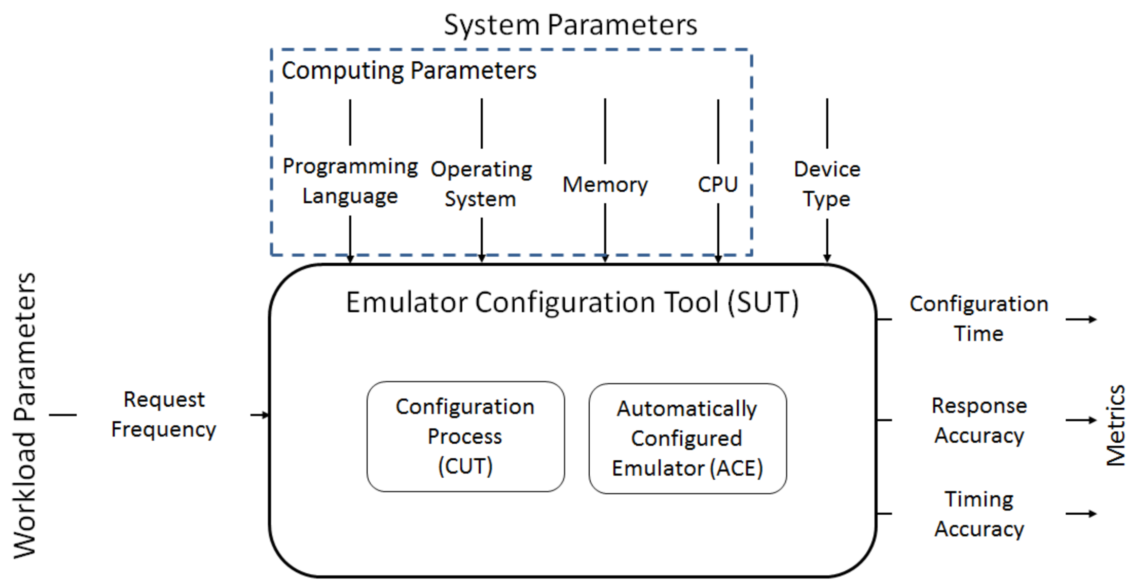


Figure 4.1: Emulation Configuration Tool

may also fail if the computing factors, such as speed, CPU, hard drive and memory, do not allow the ECT to complete execution.

4.4 Workload

The workload delivered to the system is HTTP GET requests sent to the device. For this research, this is a request to the *index.html* page of the PLC web server. This page is chosen for its size and complexity. The tool used to generate this workload is a script called *htmlget_mt*. It is a previously validated custom tool used in the Gumstix research [Jar13]. The program works by sending GET requests to the *index.html* of the device. Each request is sent on a different port. This feature allows packet streams to be separated for analysis.

The workload generator sends 200 GET requests to the page during each experiment. A set of 200 requests is selected because the *htmlget_mt* program produces a segmentation fault around 300 requests; therefore, a set of 200 requests is chosen to consistently send successful requests to the devices. Experiments are repeated 5 times for each combination of workload and system factors. This means that a total of 1000 requests are sent for each experiment, which allows for a small confidence interval for the mean.

Before any trials are run, the workload virtual machine is brought to a point in which the CPU usage is at a steady state, and there is no network usage. Between each trial, the Linux System Monitor program is used to determine a time in which the CPU usage has leveled out again and network usage has returned to zero. In addition, a wait time of one minute is placed between each experimental trial. This wait time ensures that the CPU usage and network activity are at a steady state.

4.4.1 Request Frequency.

The request frequency is the rate at which requests are sent to the CUT. Any aspect in which the ACE behavior diverges from that of the real device is an opportunity for fingerprinting. Therefore, the frequency is varied to determine response accuracy of the ACE at different levels.

In this research, there are two levels for the request frequency. The first is the *PLC Break*; it is the level at which the PLC can no longer send responses fast enough. The second level is a slow frequency at which both devices are able to easily handle requests. A breaking point for the PLC was established in the MCE Gumstix study [Jar13]. Due to changes in the experiment environment, these breaking points were reevaluated for this study.

To find the breaking point of the PLC, requests were sent to the PLC at different frequencies until the PLC cannot respond to the requests. The workload generator, *htmlget_mt*, places a built-in, changeable delay between requests. In pilot studies, requests were sent with varying delays. Using a binary search method, a range of delays were determined to be near the PLC breaking point. This range was found to be 38 milliseconds (26.32 requests/second) to 45 milliseconds (22.22 requests/second).

To determine the delay used as the *PLC Break* level for this experiment, requests are sent at set increments. The point at which the PLC responds to at least 99% of the requests is considered the *PLC Break* level. This means that the PLC is just beginning to fail to respond to requests. For every trial, 200 requests are sent to the device, and trials are repeated five times for each frequency. First, the delay is increased from 38 milliseconds to 45 milliseconds in increments of one millisecond. At a delay of 42 milliseconds, the average response success is 94.7%. To narrow in on a more accurate breaking point, sets of requests are sent at 42.5, 43.5, and 44.5 milliseconds. At a delay of 44.5 seconds (22.47 requests/second), the PLC successfully responds to an average of 99.4% of the requests. This is the *PLC Break* level. Figure 4.2 shows the percentages of successful responses at each level. The figure shows the response percentages for each trial. The line represents the average response percentage for each frequency. The resulting breaking point is shown in Table 4.1.

Table 4.1: Request Frequency. This is the rate at which requests are sent to the device.

Frequency Label	**Frequency (requests/second)**
Slow	10
PLC Break	22.47

4.5 Performance Metrics

To measure the performance of the configuration tool, the system is evaluated in terms of both the configuration process and the final ACE. The metric for the process is the time it takes to create the emulator. The time it takes to create the ACE is a combined execution time of the configuration scripts. The time for the first script begins with the execution of the wget command in the *auto-wget.py* script, and ends when the *inputFinder.py* ends

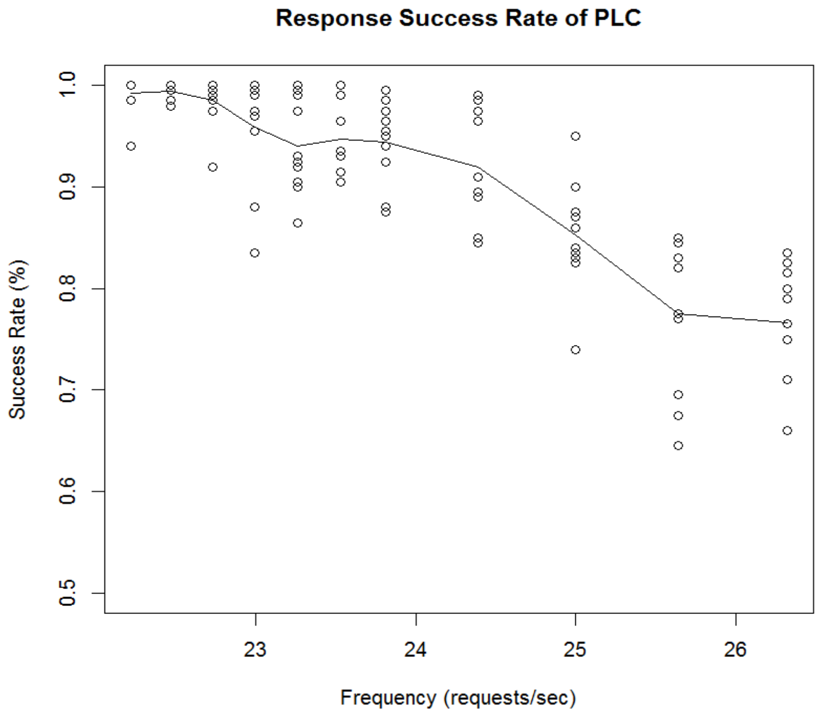


Figure 4.2: Successful Response Rate at Varying Frequencies.

execution. The time of the second script starts when *parse_pcap.py* opens the pcap file for reading. The second time ends when the *config_emulator.sh* script completes execution. If the configurator exits without creating an emulator, the time that the process exited is noted, but marked as a failure.

The ECT is evaluated, both on the performance of the configuration process, and the accuracy of the resulting emulator. Emulator accuracy is measured in terms of packet response time accuracy and the response accuracy. To maintain consistency in evaluation techniques, these metrics are based on those used in preceding research [Jar13].

To evaluate the response accuracy and the response time accuracy metrics, a set of GET requests are first sent to the *index.html* page of the real PLC at both the *Slow* and *PLC Break* frequencies. The responses to these GET requests serve as a baseline for measuring the performance of the ACE. Next, the same requests are sent to the emulator at both frequencies. A set of 200 requests are sent in each trial. Those trials are repeated five times for a total of 1000 requests to each device. Trials are performed at least 60 seconds apart to ensure no residual network traffic or CPU usage affects the next trial. In addition, the network traffic and CPU usage are monitored in Linux's System Monitor program. Trials are not run until the CPU usage has returned to the same level as it was before any trials are run and no network activity is present.

4.5.1 Response Accuracy.

Response accuracy is based on how similar the ACE's response is to that of the real PLC. This is measured in the number of bytes in the emulator's response that are the same as those in the response of the real PLC.

Using Wireshark, the traffic of each conversation is recorded into two separate pcap files. The first pcap file is a recording of the response traffic from the PLC to the workload generator. The second pcap file is a recording of the response traffic from the ACE to the workload generator. A program called *pcap_accuracy_parse* is used to compare the pcap files. This program was created and validated in the Gumstix MCE research [Jar13]. The pcap files are compared by separating the traffic into packet types. All response packets of one type from the emulator traffic are compared to the corresponding packets from the PLC traffic. This is done by comparing the packet bytes. For this research, the packets are separated into eight packet types. This is because the Koyo PLC responds to the *index.html* page in eight packets. This is based on the size of the page and the maximum segment size (MSS) used by the Koyo PLC. In future research, experiments may need to use a different number of packets to recreate an accurate comparison. A visualization of this process is shown in Figure 4.3 [Jar13]. Certain packet bytes will always be different (e.g., sequence number), and are not included in the comparison. These bytes are considered non-deterministic.

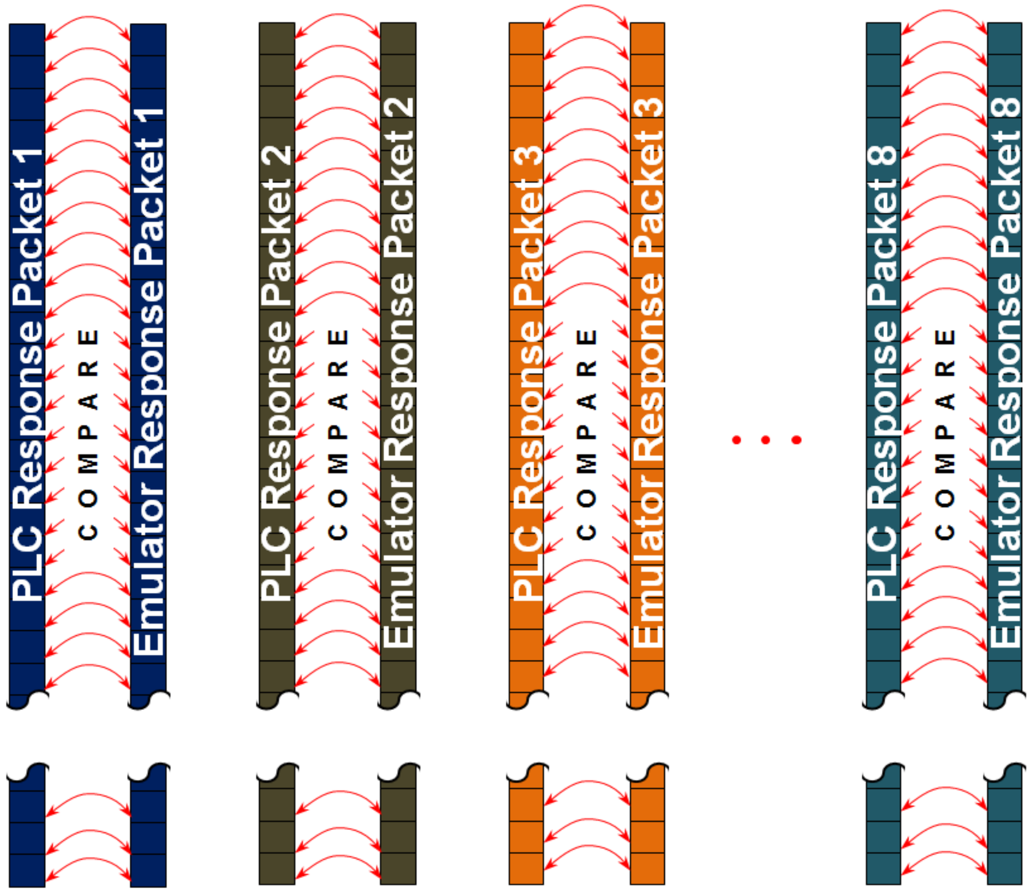


Figure 4.3: Method for Comparing Packet Bytes. [Jar13]

4.5.2 Response Time Accuracy.

The second part of accuracy measurement is response time accuracy. For this experiment, response time "starts with the first packet of the TCP handshake [from the workload generator to the emulator], and concludes with the final TCP ACK packet from the [emulator]" [Jar13]. As with the response byte accuracy measurement, a baseline for an expected average response time is created by sending requests to the PLC. A program called *web_time_parse* is used to find the response time for each request. This program is also created and validated in the Gumstix MCE research [Jar13]. When the program sends requests to the device, it sends each request on a different outgoing port. The program is written in C, and uses the *sockaddr_in* structure for port handling. Because the workload generator sent each request on a different port, *web_time_parse* is able to separate each stream based on the destination port number of the response packets. It then calculates the difference in time between the start of one stream and the start of the next stream. The response times from all 1000 requests are averaged. This is the average response time for the device. The average response time of the ACE is compared to the average time of the PLC.

4.5.3 Configuration Time.

The configuration time is defined as the total execution time of the configuration scripts. Because of the way the scripts are run, this time is calculated from the sum of the first script's run time and that of the second script. To calculate the runtime of each script the program uses Python's built in *time* module. For each script, a timer begins before the execution of any functions or calculations. The timer stops when no more calculations are running. For some scripts, user input is required at the start of the program. The timer starts after all user input is accepted. This is because that time is dependent on the user, rather than the efficiency of the code. In addition, no configuration functionality begins until all user input has been accepted.

The configuration time is measured for 25 separate instances. A sample of 25 is based on the Central Limit Theorem. A sample of this size is usually sufficient for a normal approximation [Han04]. Before each measurement, the ACE virtual machine is reverted to a snapshot. The snapshot does not contain any of the emulator files that result from the configuration. Most of the scripts create files and folders during the configuration process. An initial presence of these files would affect the time it takes to configure the emulator. Reverting to a snapshot eliminates the possibility of this effect.

4.6 System Parameters and Factors

Parameters that affect system performance include computing parameters and device type. Computing parameters the research considers include memory, CPU speed, the programming language, and the operating system of the PC. The device type is the device being evaluated: the real PLC or the ACE.

Computing Parameters. Various aspects of the computing environment can affect the speed of the configuration tool and the response of the emulator. The CPU speed and available memory have a direct impact on the time it takes for the configurator to run. The programming language can affect the runtime of the configurator. The operating system can also cause a certain amount of overhead. For this research, the configuration tool is written in Python. For this experiment the workload generator and the ACE are on separate Ubuntu 13.10 virtual machines with 20GB of disk space and 1GB of memory, and an Intel Core i7-2760QM 2.4GHz processor.

Device Type. The device type parameter is the device being evaluated. The type is either the real PLC or ACE. In this study, the PLC is a Koyo DirectLogic DL405 version 4.0.1.1735. The emulator consists of a script that is emulating the web server of the Koyo PLC and additional files used to reference response behavior.

4.7 Experimental Setup

To run the experiment, a workload generator is connected to either the PLC or the emulator. For this experiment the workload generator and the ACE run on separate Ubuntu 13.10 VMWare virtual machines, each with 20GB of disk space, and 1GB of memory. Both of these virtual machines are running on the same host with an Intel Core i7-2760QM 2.4GHz processor. The PLC is a Koyo DirectLogic DL405 version 4.0.1.1735. The workload generator uses the program *htmlget_mt* to generate and send GET requests to the main page of the emulator and the PLC. While sending the workload, the generator system is, also, running Wireshark to collect the traffic between the workload generator and the device being evaluated. A pictorial description of this setup is shown in Figure 4.4. All three machines are on the same isolated subnet. The machine running the virtual machines is connected to the PLC by a direct Ethernet connection. Both of the virtual machines' network adapters are set to the VMnet0 setting in VMWare.

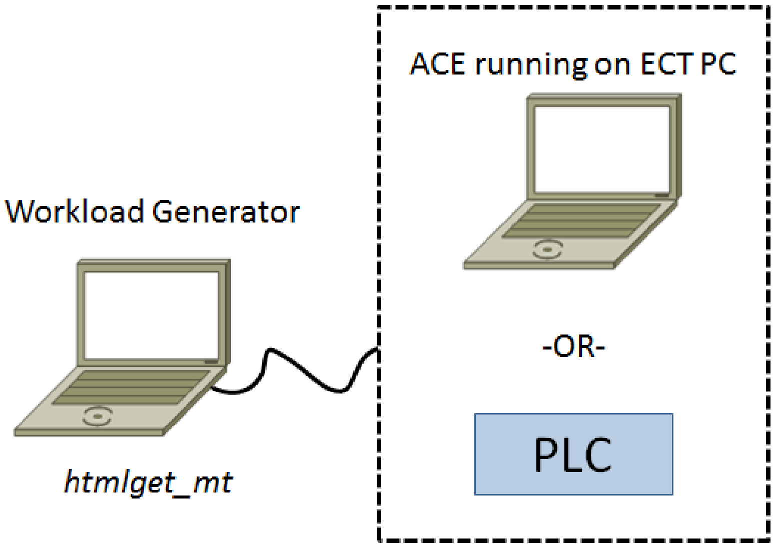


Figure 4.4: Experimental Setup

4.8 Evaluation Technique

The technique for evaluating the configuration process is a direct measurement of the configuration tool. The evaluation uses a Koyo DirectLogic 405 PLC [Koy13] as the baseline for measurement. To measure the accuracy of the emulator, this research uses measurement techniques from its preceding study [Jar13]. Because this research focuses on emulating web protocols, accuracy is measured in terms of packet byte accuracy and timing accuracy. The accuracy of the packets is based on the number of bytes that match the bytes in packets sent by the baseline device. The timing accuracy consists of the time that the emulator takes to respond after receiving a query. Using these measurements of accuracy, this study sends queries to the baseline devices and the emulator, and compares responses of each baseline with those from the emulator.

The configuration process is evaluated on configuration time. There is no current baseline for this metric. The configuration time is measured during 25 configuration replications. The average of these times will set a baseline for the configuration time of an ACE.

4.9 Experimental Design

The research has one system factor, with two levels, and one workload parameter with two levels. Running 5 replications means that the research runs 40 tests. The packets sent by the emulator should be 100% accurate in order to avoid the possibility of fingerprinting. Due to the high number of repetitions allowed by this research, a 95% confidence level for the difference in means is a goal for this study.

4.10 Methodology Summary

This research creates a tool for automatically configuring an device to emulate web protocol behavior of a Koyo DirectLogic PLC. It does so by probing the programmable logic controller (PLC) and analyzing traffic from the PLC communication, then saving necessary information into a directory, which is discussed in Section 3.2. The emulator uses that database to create a behavior profile for that particular PLC. The service that this tool provides is the generation of an emulator that mimics known web protocols for a PLC. How well the configuration tool performs, in creating the emulator, is based on the accuracy of the responses, the time it takes to respond, and the time it takes for the configuration process to complete.

To evaluate the automatically-configured emulator (ACE), it is initially tested against an actual Koyo DirectLogic PLC. If the emulator can achieve accuracy that is within a 95% confidence level, with respect to the PLC, then the ACE would be considered sufficiently accurate at evaluating the PLC.

V. Results and Analysis

This chapter discusses the results from the measurements of the emulation configuration tool (ECT). The ECT is analyzed based on the three metrics: response accuracy, timing accuracy, and configuration time.

5.1 Response Accuracy

Measurements for response accuracy are taken across all 1000 trials, but there is no change in response accuracy between trials. For both the PLC and the ACE, a total of 2444 bytes are sent during a response stream. These 2444 bytes are from the eight packets sent in response to the GET request. The eight packets are shown in Figure 5.1. The first packet is part of the TCP handshake. The second packet acknowledges the GET request was received by the device. Packets three through six contain the HTML data from the *index.html* web page. Packet seven is the HTTP OK message, telling the workload generator that the request was successful. The final packet is part of the tear-down sequence. The sum of the bytes in each packet make up the 2444 bytes in the response.

No.	Source	Protocol	Length	Info
1	10.1.0.112	TCP	60	http > commplex-main [SYN, ACK] Seq=0 Ack=1 Win=2048 Len=0 MSS=512
2	10.1.0.112	TCP	60	http > commplex-main [ACK] Seq=1 Ack=258 Win=2048 Len=0
3	10.1.0.112	TCP	566	[TCP segment of a reassembled PDU]
4	10.1.0.112	TCP	566	[TCP segment of a reassembled PDU]
5	10.1.0.112	TCP	566	[TCP segment of a reassembled PDU]
6	10.1.0.112	TCP	506	[TCP segment of a reassembled PDU]
7	10.1.0.112	HTTP	60	HTTP/1.1 200 OK (text/html)
8	10.1.0.112	TCP	60	http > commplex-main [ACK] Seq=1990 Ack=259 Win=2047 Len=0

Figure 5.1: Eight Response Packet Types

Of the 2444 bytes, only 2310 are included in the analysis of the response accuracy. The other 134 bytes are part of the non-deterministic bytes. Recall, that the non-deterministic bytes have unpredictable values and are not included in the accuracy analysis. As shown in Table 5.1, for all 1000 trials at both the *Slow* and *PLC Break* frequencies, 2259 out of 2310 deterministic bytes (97.79%) are correct. Because the byte accuracy remains the same across all trials, there is no deviation and therefore no confidence intervals.

Table 5.1: Response Accuracy of the ACE

Frequency	**Total Responses**	**Number of Correct Bytes**	**Percent Correct**
Slow	1000/1000	2259/2310	97.79%
PLC Break	981/1000	2259/2310	97.79%

The byte values of the PLC's reponse packets serve as a baseline for evaluating the accuracy of the ACE responses. A total of 51 bytes in the emulator's response packets are incorrect out of the 2310 bytes of the response. Table 5.2 shows all of the incorrect bytes, and which packet types they appear in. For instance, bytes 0x07-0x0a appear incorrectly in all eight packet types, while byte 0x47 is only incorrect in packet type three. Column three shows the total incorrect bytes that each set accounts for.

Four incorrect bytes appear in all eight packet types. These are bytes 0x07, 0x08, 0x09, and 0x0a. This accounts for 32 incorrect bytes of the total 2310 bytes. These bytes specify the middle four bytes of the six byte emulator media access control (MAC) address. The first byte of the MAC address is correct for the ACE. The last byte is part of the non-deterministic bytes, and not considered. Figure 5.2 shows the format of the Ethernet II header [Cis12]. The highlighted section is the six byte source MAC address.

Table 5.2: Incorrect Bytes in the ACE Response Packets

Byte(s)	Description	Packet Type Location	Total Incorrect Bytes
0x07, 0x08, 0x09, 0x0a	MAC Address	1,2,3,4,5,6,7,8	32
0x12, 0x13	IP ID	2,3,4,5,6,7,8	14
0x2f	TCP PSH Flag	4,5	2
0x30, 0x31	Window Size	8	2
0x47	Letter c in content-type	3	1

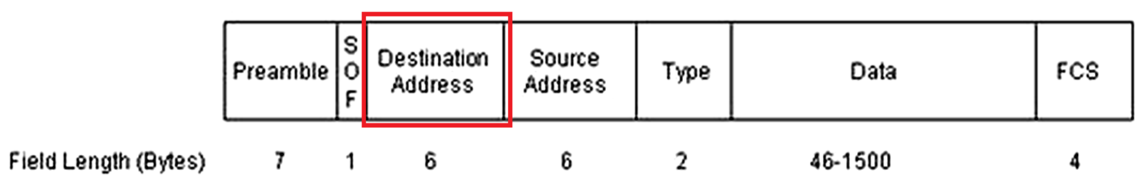


Figure 5.2: Format of the Ethernet II Header [Cis12]

Figure 5.3 shows the difference of the MAC address bytes in Wireshark. The first three bytes of the MAC address make up the organizationally unique identifier (OUI), which identifies the device vendor. The top section shows the Ethernet header of the PLC, while the bottom section shows the ACE header. As shown in the source section of Figure 5.3, Wireshark identifies the PLC as a Host Engineering device. Host Engineering is the vendor associated with Koyo PLCs. Because the MAC address of the ACE is not altered, Wireshark still identifies it as a VMWare device. Changing at least the OUI bytes to match those of the PLC would give the impression that the device is authentic.

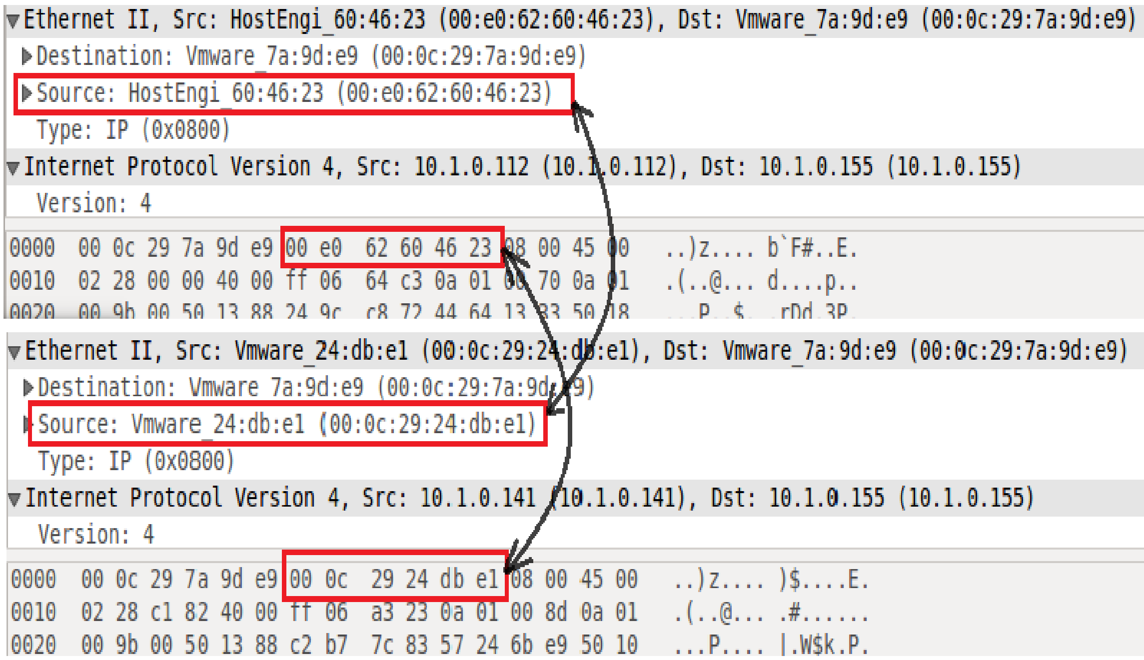


Figure 5.3: Difference in MAC Address Bytes

PLCs are commonly placed behind a router [BLK07]. This configuration could potentially mask the MAC address of the emulator. In the future, if a situation were identified that an attacker is able to get into the same subnet that the ACE honeypot is on, adding functionality to change these bytes would be possible. It would require adding an additional rule to the *config_emulator.sh* file.

The second set of incorrect bytes are bytes 0x12 and 0x13. These are the IP identification (IP ID) number of the packet. The IP ID is used to label datagrams. Therefore, if the datagram is fragmented, each fragment will have the same IP ID. Then the fragments can be reassembled [KuR13]. Figure 5.4 shows the formats of the IP and TCP headers [Hus04]. The first highlighted section shows the IP ID field.

In all packets from the PLC, the IP ID is 0x0000. The IP ID of the first packet type from the ACE is also 0x0000. After the first packet, though, the ACE sets the IP ID and increments it by one for each subsequent packet type. The implementation of the IP ID is different across operating systems, so a deviation from the PLC is a potential area for fingerprinting [Ark02].

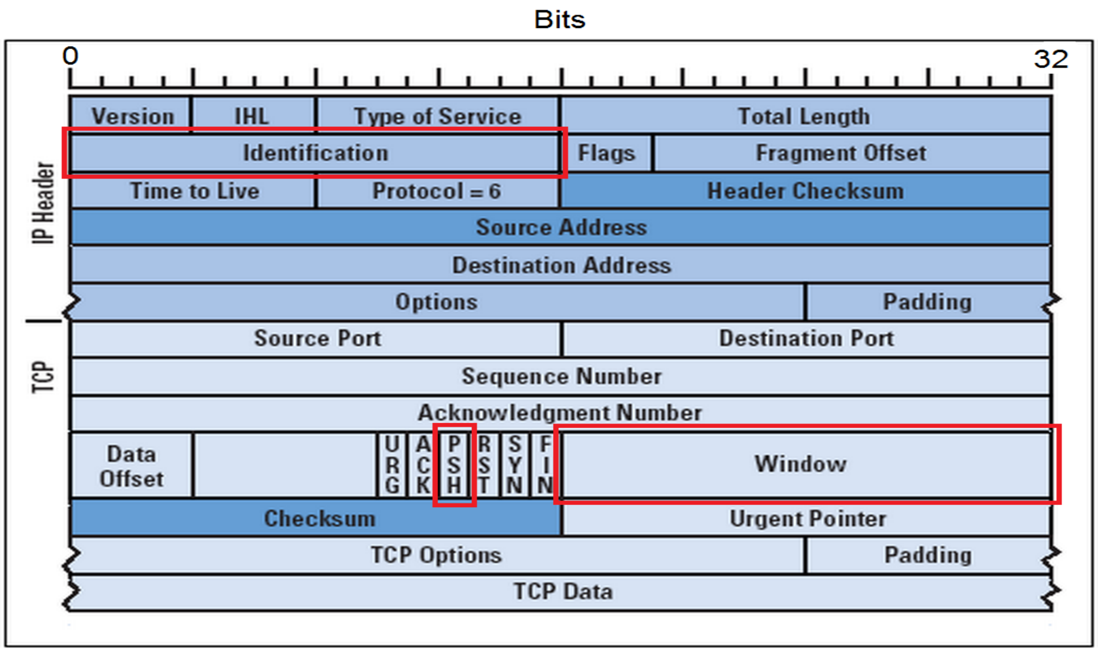


Figure 5.4: Format of the IP and TCP Headers [Hus04]

The next byte difference is byte 0x2f, which is in packet types four and five. The 0x2f byte is part of the eight bits that include the six TCP flags. The second highlighted section of Figure 5.4 shows the location of the PSH flag. In the PLC responses, the value of 0x2f byte is 0x18. The 0x2f byte in the ACE response has a value of 0x10. This difference comes from the TCP PSH flag bit, which is set for the PLC, but not set for the ACE. The PSH Flag bit, “indicates that the receiver should pass the data to the upper layer immediately” [KuR13].

The next two incorrect bytes are 0x30 and 0x31. The location of these bytes are displayed in the last higlighted field of Figure 5.4. These two bytes make up the TCP window size (RFC 1323) [JBB92]. The bytes are only incorrect in packet type eight. For the PLC the values are 0x07ff, or 2047 decimal. In the ACE the values are 0x0800, or

2048 decimal. For all other packet types, the window size for both the PLC and the ACE, is 2048 decimal. Both the PSH flag and window size are handled in kernel-space [Ste93]. Because the configuration process currently runs in user-space, changing this byte is out of the scope of this study.

The last incorrect byte appears in packet type three. This is byte 0x47, which is the letter *c* in the phrase *content-type*. This is a simple difference that arises from Python's *httplib* library. When forming the response dictionary, the ECT uses the HTTPConnection object from *httplib* to send GET requests from the emulator and record responses. The content-type section of the response header is parsed using the *getheaders* function of the HTTPConnection response object. The return value of this function contains header values as string objects. The httplib appears to disregard capitalization for the header strings. The *getheaders* function is the method for discovering the value of the header fields. The configuration process has no prior knowledge of whether or not the field values are capitalized. If future research determines that the *c* is capitalized for all PLC brands, then this can be hard-coded into the configuration process. Figure 5.5 shows the difference between the PLC and the ACE for byte 0x47.

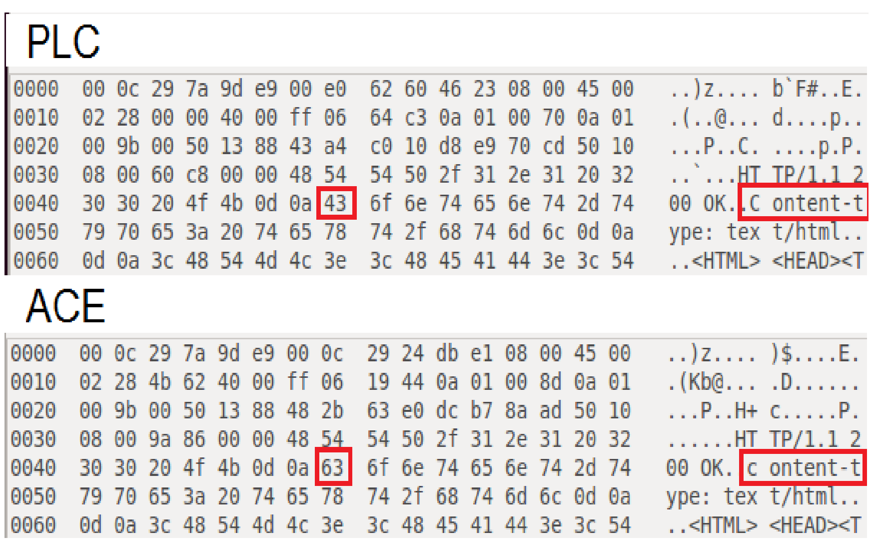


Figure 5.5: Comparison of *content-type* Phrase

The RFC states that the field names of the HTTP headers are case-insensitive (RFC 2616) [Fie99]. In addition, the byte difference is syntactical rather than semantical, so the difference is not critical. A skilled user may take note of the difference, but the change would not likely throw a red flag.

In the Gumstix manually-configured emulator (MCE) research, the byte accuracy of the emulator was 99.79% [Jar13]. An accuracy percentage, closer to that level, would have further shown the validity of automatic configuration. Some of the incorrect bytes in the ACE are the same incorrect bytes in the MCE. Specifically, this includes bytes 0x2f, 0x30, and 0x31, which were incorrect in the same packet types as the MCE. Thus, the largest cause for the deviation in the ACE accuracy percentage are the four MAC address bytes.

5.2 Timing Accuracy

Table 5.3 shows a summary of the response times for each device at the different frequency levels. The mean response times across 1000 trials are shown, along with the standard deviation and 95% confidence intervals for each. For the *PLC Break* level, two summaries are shown for the PLC. The first shows the summary of all data collected. The original data show that five outliers with p-values of less than $1x10^{-20}$ exist. Because the outliers were so extreme, they were removed to more easily show the relationship between the PLC and the ACE response times. The summary of that data is shown in the last line of Table 5.3, labeled PLC (N.O.).

5.2.1 Slow.

The average response times from the *Slow* level, shows the behavior of each device at a request frequency that both the PLC and the ACE can easily handle. From Table 5.3, the average response time of ACE is 1.37 times faster than that of the PLC. Although the difference is not huge, the 95% confidence intervals do not overlap. In addition, a

Table 5.3: Summary of Device Response Times

Frequency	Setting	Total Responses	Mean Response Time (s)	S.D.	95% C.I.
Slow 10 Req/sec	ACE	1000/1000	0.0295714	0.002653	0.0294-0.0297
	PLC	1000/1000	0.0405504	0.001297	0.0405-0.0406
PLC Break 22.47 Req/sec	ACE	1000/1000	0.0381775	0.003843	0.0379-0.0384
	PLC	989/1000	0.0562203	0.136387	0.0477-0.0647
	PLC (N.O.)	984/1000	0.0509099	0.016059	0.0499-0.0519

two-sided t-test shows that the means are different with a p-value of $2.2x10^{-16}$. Thus, the average response time of the ACE is statistically faster than that of the PLC.

Figure 5.6 shows a scatter plot of the response times from each trial. For visualization purposes, the scatter plots in Figure 5.6 and Figure 5.7 (discussed later) show the data with first order outliers removed. It is easy to see that, at the *Slow* frequency, the response times of the PLC are slower than those of the ACE.

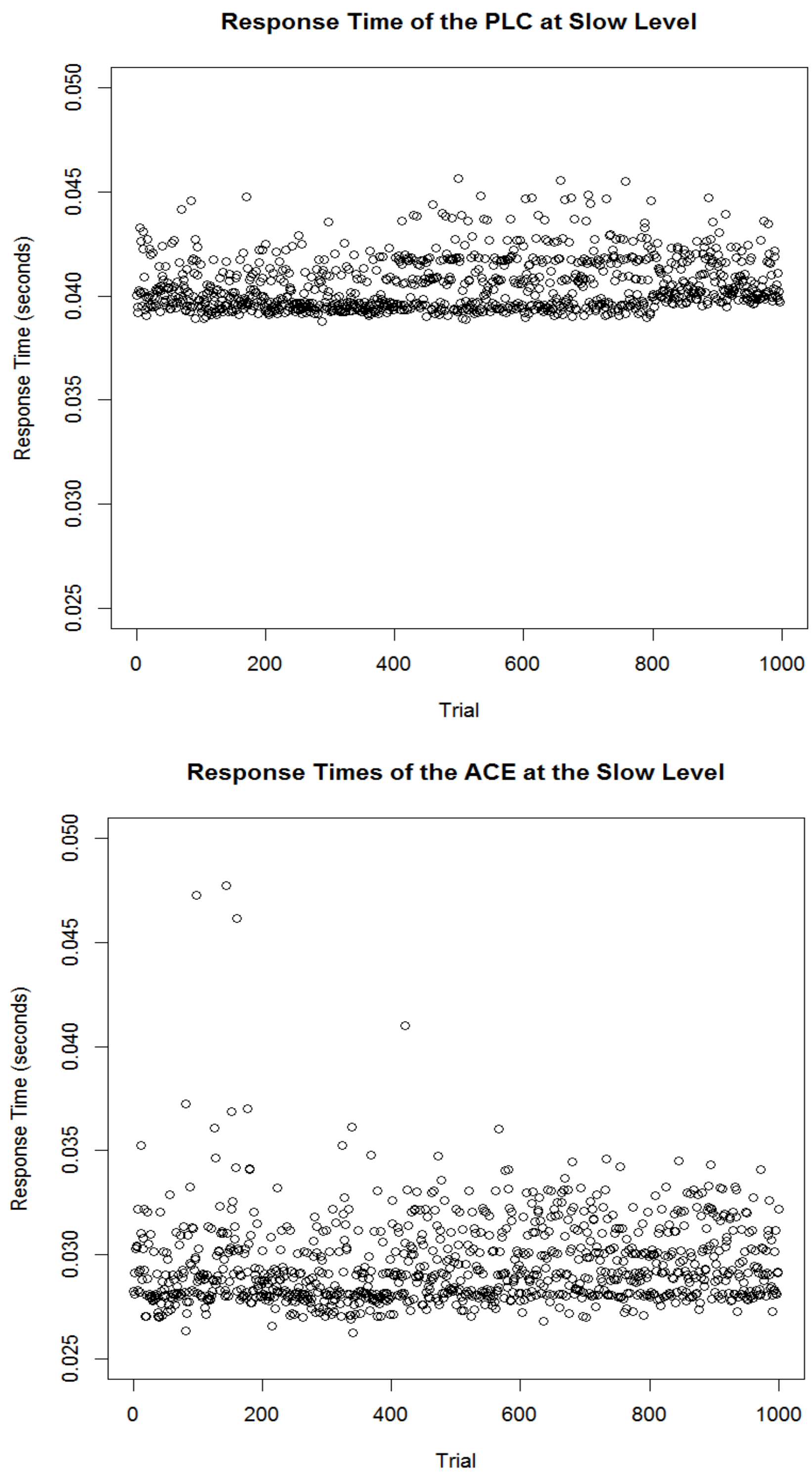


Figure 5.6: Device Response Times at the Slow Level.

5.2.2 PLC Break.

This section discusses the performance of the devices at the *PLC Break* level. This is the level at which the PLC just begins failing to respond to requests. Table 5.3 shows that, again, the ACE responds faster than the PLC. On average, the ACE responds 1.47 times faster than the PLC. In addition, a two-sided t-test for the means at the *PLC Break* level also shows that the means are different with a p-value of $2.2x10^{-16}$. Figure 5.7 shows the response times for each device with the first order outliers removed.

From these graphs, we can see that a number of outliers still exist for the PLC response times. Because this frequency is the breaking point of the PLC, outliers are likely to occur. The PLC only responded to 989 of the 1000 requests. This means that the requests timed out for the other 11 attempts. It is reasonable to have variability in the responses. Because all data points are valid, an analysis of all response times must be included. Excluding the extreme outliers, though, allows for a more expressive view of the means. The final line of Table 5.3 shows a summary of the PLC response times without the outliers. The standard deviation is smaller, but it is still much larger than the standard deviation of the ACE response times. Additionally, even though the average of the no-outlier data is lower, the 95% CI still does not overlap that of the ACE response times.

Figure 5.8 shows another view of the response times. In this figure, the graph of the ACE response times still shows the data with only the first order outliers removed. The PLC graph, on the other hand, shows the data with all outliers removed that have a p-value of less than 0.05. These graphs show that there is still a relatively larger spread amongst the PLC data. It is now easier to see that even the fastest response times of the PLC are still slower than those of the ACE.

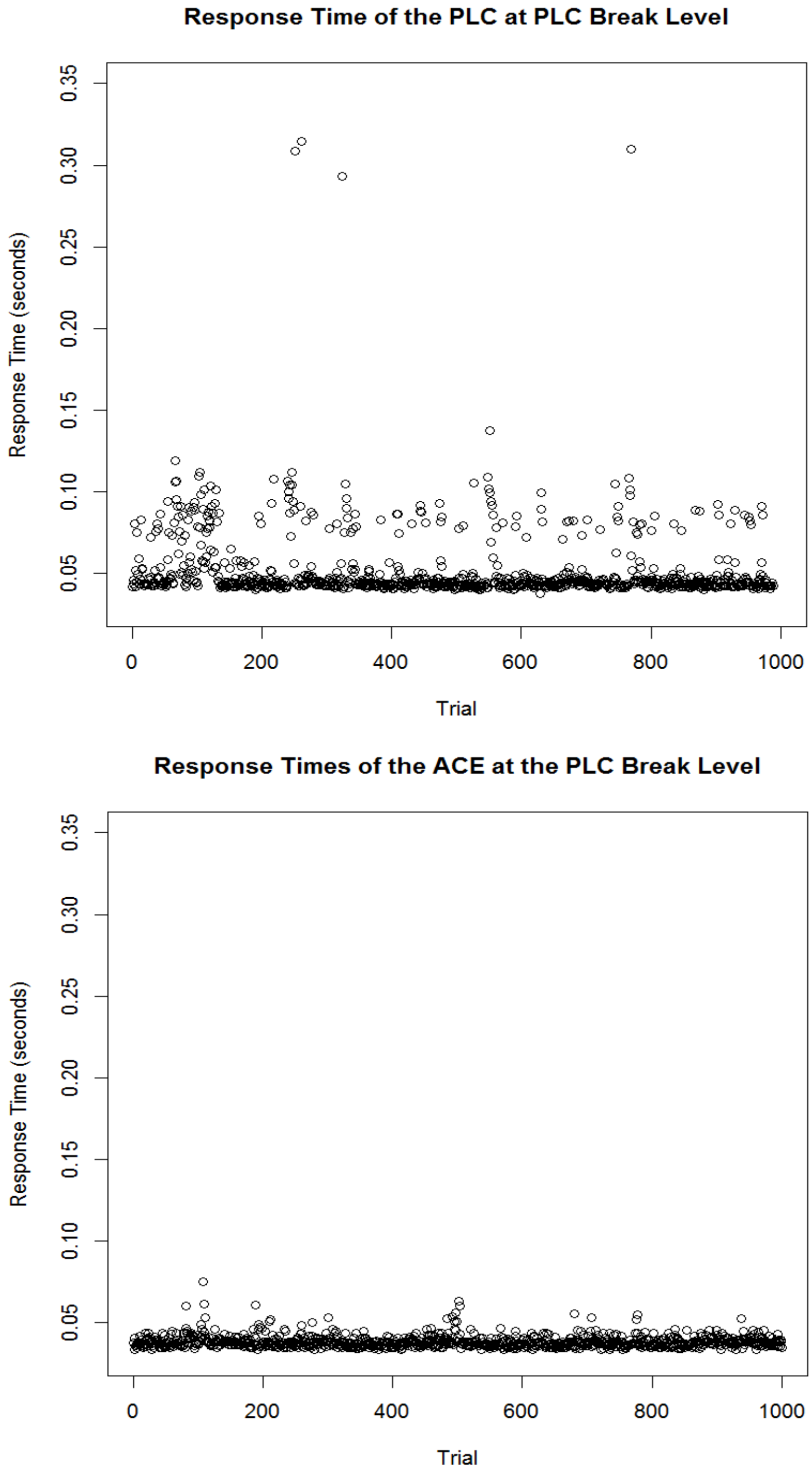


Figure 5.7: Device Response Times at the PLC Break Level

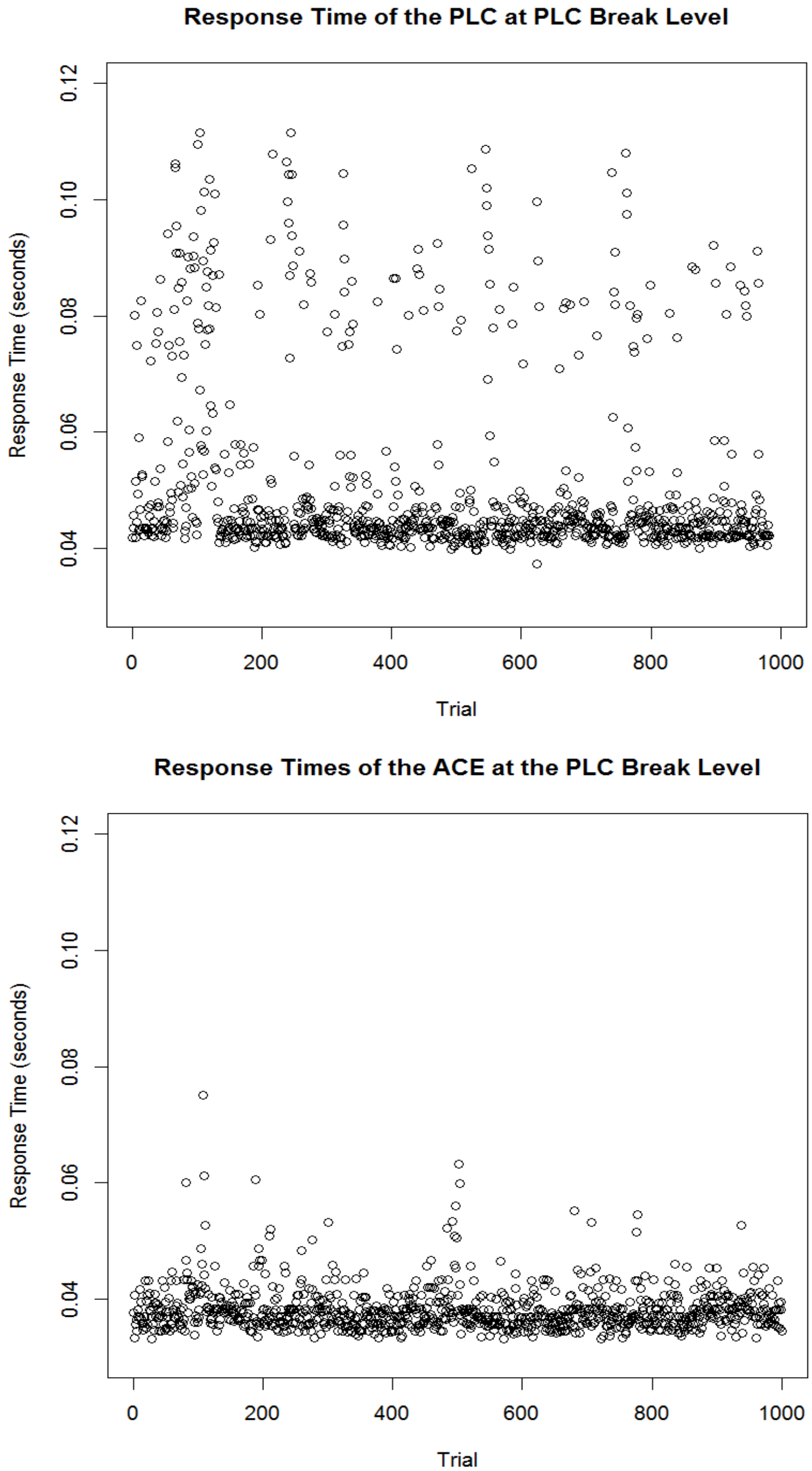


Figure 5.8: Device Response Times at the PLC Break Level. All outliers for the PLC response times have been removed.

5.2.3 Adding an Artificial Delay.

The results above show that the response times of the ACE are faster than those of the PLC. This difference could be lessened by adding an artificial delay to the ACE responses. Pilot studies suggested that a delay of 1.5 ms could be suitable to lessen the gap between the ACE and PLC response times. Initial considerations for maintaining autonomy and minimizing complexity concluded that the static delay is potentially an acceptable solution. Table 5.4 shows the response times of the ACE without the delay, alongside those with an added 1.5 ms delay. The delay does help in decreasing the gap between response times.

Table 5.4: Comparison of Response Times for the ACE With the Delay and Without the Delay

Device	**Request Frequency**	**Response (Seconds)**		**PLC Average**	**Deviation from PLC Average**	
		Mean	**S.D.**			
ACE	Slow	0.029571	0.002653	0.040550	0.0110	1.37 x Faster
ACE w/delay		0.031071			0.0095	1.31 x Faster
ACE	PLC Break	0.038178	0.003843	0.056220	0.0180	1.47 x Faster
ACE w/delay		0.039678			0.0165	1.42 x Faster

A more accurate delay could be created by learning response time information during the configuration process. Part of the configuration process includes sending GET requests to each the PLC. Packets captured from this interaction with the PLC are analyzed. Using this traffic configuration tool could determine the response times for each page request. The average response times could be added to the ACE as an artificial delay. This could be a static delay, like the one used in this study. It could also be a varying delay withing a range near the PLC average. Because the PLC response times have a large amount of variance, adding a variable delay, would likely create more realistic response times.

5.2.4 Comparison to Gumtstix MCE Research.

This section provides comparative results of the Gumstix MCE experiments to those of the ACE. Table 5.5 shows a summary of the averages from each study. The lines labeled *MCE PC* are the results of the MCE running on an Ubuntu 2.6.35 PC [Jar13]. The average PLC times from the MCE study are also from a Koyo DirectLogic PLC. Both the ACE and MCE use Python's web server library as the base for the emulator. It is important to take into consideration the tests of the MCE and the ACE were run in different testing environments. In addition, the *PLC Break* frequency that was derived for the MCE research was 27.49 requests/second. The *PLC Break* frequency in this research is 22.47 requests/second.

For the *Slow* frequency, the average response time of the MCE is 13.4 times faster than its respective PLC average. The ACE is 1.3 times faster than the PLC average in this research. The average MCE response time at the PLC Break frequency is 38.4 times faster than the PLC average. The average time of the ACE is 1.4 times faster. In addition, the standard deviation of the MCE response time is smaller than that of the ACE at both frequency results.

Table 5.5: Comparison of Response Times for the ACE and the Gumstix MCE [Jar13]

Device	Request Frequency	Total Responses	Response (seconds)		PLC Average	Deviation from PLC Average	
			Mean	S.D.			
MCE PC	Slow	1000	0.002817	6.26e-4	0.037685	0.0349	13.4 x Faster
ACE		1000	0.031071	0.002653	0.040550	0.0095	1.3 x Faster
MCE PC	PLC Break	1000	0.002449	6.58e-4	0.093950	0.0915	38.4 x Faster
ACE		1000	0.039678	0.003843	0.0562203	0.0165	1.4 x Faster

5.3 Configuration Time

This research creates a standard configuration time for the ACE. This metric is not compared to any baseline. Also note that the configuration time is calculated in only one computing environment. Changes in processor speed and memory could greatly affect the runtime. To determine an average, the ACE is configured 25 times. The total run time of all configuration programs is recorded for each repetition. The average configuration time for the ACE is 9.8 seconds with a standard deviation of 0.659 seconds. A histogram of the configuration times is shown in Figure 5.9.

This study is 95% confident that the average configuration time is between 9.54 and 10.06 seconds. This confidence interval is under the five minute threshold value. An average time of 9.8 seconds is acceptable for the intended implementations of the ECT. This is, also, within reason for the amount of work accomplished by the tool.

5.4 Summary

This chapter presents the results of the ECT experiments. Testing of the response accuracy shows that the ACE matches the PLC across 97.79% of the packets within the

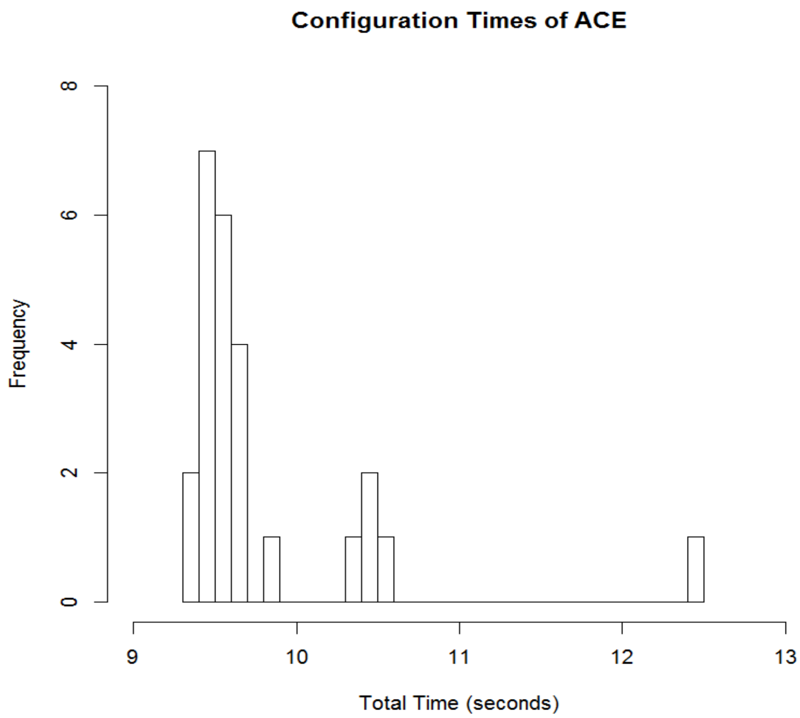


Figure 5.9: Histogram of the Configuration Time of the ACE Across 25 Replications

response. While this is less than 100%, it shows that a high level of accuracy can be achieved, even with an automated process. Measurements of response time accuracy shows deviations from the PLC average, but the response times are an improvement from the previous MCE research. The configuration time executed in 9.8 seconds on average. The low execution time shows that the configuration process does not require large computing resources. Finally, the configuration time allows for expansion of options and protocols configured by the ECT.

VI. Conclusions

It has been shown that critical infrastructure is at risk for network attacks. Being able to successfully mitigate these attacks requires knowledge about the vulnerability in ICS systems and the methods that attackers use to gain access. It is possible to use real PLCs to gain this knowledge, but PLCs are expensive. For instance, in an education environment, students may deploy real attacks in order to learn about PLC vulnerabilities. Using real PLCs for that environment could result in costly damage to the devices.

Hand-configured emulators are costly because they require the time of a skilled programmer to configure the emulator to each desired PLC. Automatic emulators offer a solution because they are inexpensive, and if they become damaged, they are easily restarted. An emulator, that can be automatically configured, not only eliminates the need for a programmer, but it also allows for a quick configuration of PLCs that look like any brand or type.

This research created a process for automatically configuring a device to emulate the web protocols of a PLC. The process gathers information about a PLC based on its response behavior and network settings. It then uses a generic web server that combines response headers and web page contents to form a response that is in line with the PLC. The resulting emulator shows promising accuracy, suggesting that the process is valid, and automatically configuring PLC emulators is possible.

Because the web interface can be the first view an attacker receives of a PLC, an accurate emulation of the web protocols is critical in creating a useful tool. Furthermore, the methods in this research can be expanded to include configuration of the industrial protocols. This would create an even more effective emulator. More discussion of future research is included in Section 6.2.

6.1 Research Conclusions

Response Accuracy. The response accuracy for this study is based on the number of correct bytes in the response packets of the ACE. The accuracy in this study was 97.79% across all tests. Because there is no variance, there are no confidence intervals for the mean. One set of incorrect bytes, that greatly affected the accuracy percentage, were the four bytes within the hardware address of the emulator. These bytes decreased the accuracy by 1.4%. Correcting these bytes would bring the accuracy of the emulator closer to the desired level. The presence of any incorrect bytes could be an area for fingerprinting, so the accuracy of the ACE could still be improved. Despite the current discrepancies, this study shows strong evidence, that it is possible to emulate the web protocols of a PLC through an automated process.

Timing Accuracy. The second metric in the research is the timing accuracy of the responses. The metric is determined by averaging the response times of the ACE for 1000 GET requests. The same requests are sent to the PLC, and the averages are compared. For both the *Slow* and *PLC Break* frequencies, the ACE responds faster than the PLC. The ACE responds 0.0095 seconds faster at the *Slow* frequency, and 0.0165 seconds faster at the *PLC Break* frequency. If the ACE is connected to an industrial network, delays from other networking devices could mask such a small difference.

Configuration Time. The configuration time metric is a measurement of the total time to run the configuration process. The threshold for an acceptable time is based on the intended implementations of the ACE. The implementations suggested by this study include educational environments and industrial network deployment. Based on these scenarios, a maximum configuration time of five minutes is considered acceptable. Including the configuration time the ACE also provides a reference point for future configuration processes. The results of this research show that the average configuration time is 9.8 seconds with a 95% confidence interval of 9.54-10.06 seconds. This confidence interval is below the five minute threshold; therefore, it is considered acceptable.

6.2 Future Work

This section discusses recommendations for future research. Research opportunities could mean expanding the functionality of the ACE, increasing its accuracy, or creating more comprehensive methods of testing.

6.2.1 Adding Functionality.

Testing ECT on other PLCs. One aim of this research is to provide evidence that an automatically configured emulator can behave like a PLC with minimal prior knowledge of the PLC. The high level of accuracy of the ACE in this study shows that this is possible. In addition, pilot studies were conducted using the ECT to configure the ACE for an Allen-Bradley Logix5555. These studies suggest that an emulator of that device could be created. Future research could use the ECT to create emulations of other brands of PLCs. Testing the accuracy of those emulators would further confirm the validity of the ECT.

Multiple Emulators. Another avenue, for increasing functionality, is using the ECT to create multiple emulators on a single device. Honeyd has shown the capability to do this with other emulators [Hon12]. Therefore, it may be possible to incorporate the ECT into Honeyd. Creating a network of virtual machines (VMs), each with their own ACE, could be another option for investigation.

Emulation of Industrial Protocols. This research focused on emulating open web protocols. To fully emulate a PLC, emulation of the PLC-specific protocols would need to be included. The largest constraint of this research would be the proprietary nature of many PLC protocols. One step in creating this emulation may be to devise an algorithm to learn the protocol well enough that it can be emulated. At that point, the algorithm could be added to the ECT. Creating a reverse engineering algorithm would not only aid in the creation of an emulator, but could be used to find vulnerabilities in PLCs.

6.2.2 Increasing Accuracy.

Many areas exist to make the ACE a more accurate emulation. One area is adding the ability to make changes to values within the web interface. For the Koyo PLC, this includes values such as the IP address of the PLC, the device name, and other settings. The current state of the ACE allows a user to make changes, but those changes are not properly updated across the entire web server. The Koyo propagates these changes through scripting. A page that indicates that the change is successful appears on the screen. Because this page only appears when a change has been submitted to a field, the ECT does not see this page during configuration. When a change is submitted to the ACE, it logs the change in a reference dictionary, but it does not display the completion page. It does not properly route back to the *index.html* page, as the Koyo would. Future research could create an algorithm for mimicking any scripting performed by the PLC. Because each PLC changes values differently, this issue may not be as prominent for other brands.

6.2.3 Accuracy Testing.

The measurements of accuracy, for this research, is based only on the byte accuracy of the response to the *index.html* page, and the timing of those responses. A more rigorous testing method would allow researchers to see potential vectors for fingerprinting the emulator. This would give researchers a list of faults that could potentially increase the accuracy of the emulator. One option, for further accuracy testing, is putting the ACE on a real network and allowing interaction with human users. This could be done with PLC experts to see if they are able to detect the difference between the ACE and the real PLC. Another option, would be to place the ACE on an Internet-facing network for outside users to interact with. The users could give valuable input in determining the ACE's ability to deceive an attacker.

Appendix A: Configuring the Emulator

The ECT has only been tested on an Ubuntu 13.10 virtual machine with 20GB of disk space, and 1GB of memory. This setup should provide all necessary system requirements. Any additional requirements are included in the generic_emulator folder. These requirements include the Python libraries used by the ECT, the configuration scripts, and the generic web server.

To run the configuration process, open two terminals in the Ubuntu machine. In the first terminal, run the following command within the generic_emulator folder:

```
$sudo tcpdump -i eth0 -w settings.pcap
```

The tcpdump program captures the traffic between configuration machine and the PLC. This traffic is saved to the settings.pcap file, which is parsed by the parse_pcap.py script. While tcpdump is running, start the configuration scripts. In the other terminal window run this command:

```
$sudo python ./auto-wget.py
```

This command starts the auto-wget script. The auto-wget program also runs the inputFinder script within the code. These scripts accomplish three tasks: (1) gather all available pages from the web server, (2) send a GET request to each page, and (3) record the PLC's response to the GET request.

Stop tcpdump, and back in the second terminal where the auto-get command was executed, run these commands:

```
$sudo python ./parse_pcap.py settings.pcap
$cd web_server
$sudo python ./webserver.py
```

The parse_pcap script looks through the settings.pcap file for a packet with the SYN and ACK flags set. This packet contains the values for the PLC network configuration options. parse_pcap extracts these values and sends them the config_emulator script. This script changes the network configuration of the emulator VM, to match those values.

Finally the webserver.py command starts the ACE. The webserver.py program is a Python based web server, with the functionality to reference other files in order to formulate its responses to GET requests.

Appendix B: Non-deterministic Fields

B.1 Non-deterministic Fields in Web Response

The information in this Appendix was created in the Gumstix MCE study [Jar13]. The Appendix is included in this paper for reference.

Table B.1 Ethernet Header Fields [Jar13]

Offset(Hex)	Description	Response Packet (1-8)
0x0B	Ethernet Address (MAC) 6th byte	1,2,3,4,5,6,7,8

Table B.2: IP Header Fields [Jar13]

Offset(Hex)	Description	Response Packet (1-8)
0x18	IP CRC 1st byte	1,2,3,4,5,6,7,8
0x19	IP CRC 2nd byte	1,2,3,4,5,6,7,8
0x1D	IP Source byte 4	1,2,3,4,5,6,7,8

Table B.3 TCP Header Fields [Jar13]

Offset(Hex)	Description	Response Packet (1-8)
0x24	TCP Destination Port byte 1	1,2,3,4,5,6,7,8
0x25	TCP Destination Port byte 2	1,2,3,4,5,6,7,8
0x26	TCP Sequence Number byte 1	1,2,3,4,5,6,7,8
0x27	TCP Sequence Number byte 2	1,2,3,4,5,6,7,8
0x28	TCP Sequence Number byte 3	1,2,3,4,5,6,7,8
0x29	TCP Sequence Number byte 4	1,2,3,4,5,6,7,8
0x2a	TCP Acknowledgement Number byte 1	1,2,3,4,5,6,7,8
0x2b	TCP Acknowledgement Number byte 2	1,2,3,4,5,6,7,8
0x2c	TCP Acknowledgement Number byte 3	1,2,3,4,5,6,7,8
0x2d	TCP Acknowledgement Number byte 4	1,2,3,4,5,6,7,8
0x32	TCP CRC byte 1	1,2,3,4,5,6,7,8
0x33	TCP CRC byte 2	1,2,3,4,5,6,7,8

Appendix C: Commands run by *config_emulator.sh*

The following appendix lists the commands run by config_emulator.sh that use values from PLC traffic packets. The values are changed on the emulator virtual machine to reflect the values that are on the PLC.

Setting	**Description**	**Command**
MTU	This is the maximum segment size (MSS) plus 40 bytes for the TCP/IP header. The MSS is the maximum amount of data that can be put in a segment.	ifconfig mtu [value]
TCP Sack	TCP selective acknowledgment. This allows the sender to specify whether or not to tell the receiver which packets have already been sent. Thus the receiver can re-transmit lost packets. [RFC 2018]	echo “[value]” >>/proc/sys/net/ipv4/tcp_sack
TCP Window Scaling	This option allows windows larger than 65KB if it is turned on. [RFC 1323]	echo “[value]” >>/proc/sys/net/ipv4/tcp_window_scaling
TCP timestamp	This specifies whether or not to calculate server uptime into the timestamp value placed in the TCP options section. If this option is turned on, it means “do not use system uptime in calculation,” otherwise use it. [RFC 1323]	echo “[value]” >>/proc/sys/net/ipv4/tcp_timestamps
TTL	The number of hops allotted to a piece of data. After the hops reach zero, the data is discarded.	echo “[value]” >>/proc/sys/net/ipv4/ip_default_ttl

Bibliography

[Ark02] O. Arkin, *A Crash Course with Linux Kernel 2.4.x, IP ID Values & RFC 791*, http://ofirarkin.files.wordpress.com/2008/11/ofirarkin2002-02.pdf, April 13, 2002, Last accessed: May 16, 2014.

[BLK07] E. J. Byres, D. Leversage and N. Kube, "Security Incidents and Trends in SCADA and Process Industries," in *The Industrial Ethernet Book*, Schondorf, Germany, Vol. 39, No. 2, http://www.iebmedia.com/index.php?id=5487&parentid=63&themeid=255&hft=39&showdetail=true&bb=1 , May 2007, Last Accessed: May 29, 2014.

[Bol00] W. Bolton, "Programmable Logic Controllers," in *Programmable Logic Controllers* (2nd Edition), Newnes, Woburn, Ch. 1, Sec. 1-3, 2000.

[ByL04] E. J. Byres and J. Lowe, "The Myths and Facts Behind Cyber Security Risks for Industrial Control Systems," *VDE 2004 Congress*, Berlin, Germany, October 2004, pp. 1-7.

[Cis12] Cisco, *NetFlow Configuration Guide*, *Cisco IOS Release 15M&T*, http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/netflow/configuration/15-mt/nf-15-mt-book.pdf, 2012, Last accessed: May 18, 2014.

[Cpn11] Centre for the Protection of National Infrastructure, *Securing the Move to IP-Based SCADA Networks*, http://www.cpni.gov.uk/Documents/Publications/2011/2011034-scada-securing_the_move_to_ipbased_scada_plc_networks_gpg.pdf, November 2011, Last Accessed: May 29, 2014.

[DHS13] U.S. Department of Homeland Security Office of the Inspector General, *DHS Can Make Improvements to Secure Industrial Control Systems* Washington, http://www.oig.dhs.gov/assets/Mgmt/2013/OIG_13-39_Feb13.pdf, 2013, Last Accessed: May 29, 2014.

[Dig11] Digital Bond, "SCADA Honeynet," http://www.digitalbond.com/tools/scada-honeynet/, 2011, Last accessed: May 18, 2014.

[Fie99] R. Fielding, et al, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616. Internet Engineering Task Force, ftp://www.ietf.org/rfc/rfc2616.txt, June 1999, Last accessed: May 23, 2014.

[FMC11] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier," Symantec, http://www.symantec.com/content/en/us/enterprise/media/security_response

/whitepapers/w32_stuxnet_dossier.pdf , February 2011, Last accessed: May 18, 2014.

[Gum12] Gumstix, www.gumstix.com, 2012, Last accessed: May 18, 2014.

[Han04] W. C. Haneberg, "Illustrating the Central Limit Theorem," in *Computational Geosciences with Mathematica*, Vol 1. Berlin, Germany: Springer, Ch. 4, Sec. 10, 2004, pp. 163-167.

[Hon12] Honeyd, http://www.honeyd.org/index.php, 2012, Last accessed: May 18, 2014.

[HoW05] J. van der Hoeven and H. van Wijngaarden, "Modular Emulation as a Long-term Preservation Strategy for Digital Objects," in *5th International Web Archiving Workshop*, The Hague, The Netherlands, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.2100&rep=rep1&type=pdf, 2005, Last accessed: May 27, 2014.

[Hus04] G. Huston, "Anatomy: A Look Inside Network Address Translators," *The Internet Protocol Journal*, Vol. 7, No. 3, pp. 3, http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-3/ipj_7-3.pdf , September 2004, Last accessed: May 18, 2014.

[Jac08] H. Jack, *Automating Manufacturing Systems with PLCs*, Hugh Jack, Allendale, Michigan, https://ia601206.us.archive.org/3/items/ost-engineering-plcbook5_1/plcbook5_1.pdf, 2008, Last Accessed: May 29, 2014.

[Jar13] R. Jaromin, "Emulation of Industrial Control Field Device Protocols," M.S. thesis, GCO, AFIT, Wright Patterson AFB, OH, http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA582482, 2013, Last Accessed: May 30, 2014.

[JBB92] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc1323.txt, May 1992, Last accessed: May 19, 2014.

[Jul91] D. Julin, et al, "Generalized Emulation Services for Mach 3.0 Overview, Experiences and Current Status," *Usenix Mach Symposium*, pp. 13-26, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.1907&rep=rep1&type=pdf, November 6, 1991, Last Accessed: May, 29, 2014.

[KLK09] D. Kang, J. Lee, S. Kim, et al, "Analysis on Cyber Threats to SCADA Systems," *Transmission & Distribution Conference & Exposition: Asia and Pacific*, Seoul, South Korea, October 2009, pp.1-4.

[Koy13] Koyo Electronics Industries Ltd.: Products, http://www.koyoele.co.jp/english/product/plc/, 2013, Last accessed: May 18, 2014.

[KuR13] J. Kurose, K. Ross, *Computer Networking: A Top-Down Approach*, Pearson Education Inc., New Jersey, 2013, pp. 235.

[Lev11] E. P. Leverett, "Quantitatively Assessing and Visualising Industrial System Attack Surfaces," MPhil thesis, CS, University of Cambridge, United Kingdom, https://www.cl.cam.ac.uk/~fms27/papers/2011-Leverett-industrial.pdf, 2011, Last Accessed: May 30, 2014.

[Nma12] Nmap Port Scanning Techniques, http://nmap.org/book/man-port-scanning-techniques.html, 2012, Last accessed: May 18, 2014.

[ObK06] O. Oberhide and M. Kair, "Honeyd Detection Via Packet Fragmentation," Networking Research and Development, Ann Arbor, MI, http://www.merit.edu/research/pdf/2006/MTR-2006-01.pdf, 2006, Last Accessed: May 29, 2014.

[PoF05] V. Pothamsetty and M. Franz, "Scada Honeynet Project: Building Honeypots for Industrial Networks," http://scadahoneynet.sourceforge.net/, 2005, Last accessed: May 19, 2014.

[PrH08] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*, Addison-Wesley, Boston, MA, Ch. 1-5, 2008, pp. 1-162.

[Pyt14] Python Software Foundation, BaseHTTPServer, https://docs.python.org/2/library/basehttpserver.html, 2014, Last accessed: May 16, 2014.

[Roh96] P. Rohner, *Automation with Programmable Logic Controllers*, University of New South Wales Press, Sydney, Australia, 1996.

[SJK13] K. Stouffer, J. Falco, and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security," *NIST SP 800-82 (Rev. 1)*, http://csrc.nist.gov/publications/nistpubs/800-82/SP800-82-final.pdf, 2013.

[Sla03] J. Slats, "Emulation: Context and Current Status," Digital Preservation Testbed, The Hague, The Netherlands, http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=6008E32930A04C717C34832A5319D8F7?doi=10.1.1.132.5566&rep=rep1&type=pdf, 2003, Last accessed: May 27, 2014

[Spi03] L. Spitzner, *Honeypots: Tracking Hackers*, Addison-Wesley, Reading, MA, 2003.

[Ste93] R. Stevens, *TCP/ IP illustrated (Vol. 1): The Protocols*, Ch. 20, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, 1993.

[Wad11] S. Wade, "SCADA Honeynets: The Attractiveness of Honeypots as Critical Infrastructure Security Tools for the Detection and Analysis of Advanced Threats," http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3130&context=etd, 2011, Last Accessed: May 29, 2014.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
19–06–2014	Master's Thesis	Oct 2012–June 2014

4. TITLE AND SUBTITLE

Toward Automating Web Protocol Configuration for a Programmable Logic Controller Emulator

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Fink, Deanna R., Civilian

5d. PROJECT NUMBER

14G216C

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-T-14-J-4

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Homeland Security ICS-CERT POC: Neil Hershfield, DHS ICS-CERT Technical Lead ATTN: NPPD/CS&C/NCSD/US-CERT Mailstop: 0635, 245 Murray Lane, SW, Bldg 410, Washington, DC 20528 Email: ics-cert@dhs.gov phone: 1-877-776-7585

10. SPONSOR/MONITOR'S ACRONYM(S)

DHS ICS_CERT

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Industrial Control Systems (ICS) remain vulnerable through attack vectors that exist within programmable logic controllers (PLC). PLC emulators used as honeypots can provide insight into these vulnerabilities. Honeypots can sometimes deter attackers from real devices and log activity. A variety of PLC emulators exist, but require manual configuration to change their PLC profile. This limits their flexibility for deployment. An automated process for configuring PLC emulators can open the door for emulation of many types of PLCs.
This study investigates the feasibility of creating such a process. The research creates an automated process for configuring the web protocols of a Koyo DirectLogic PLC. The configuration process is a software program that collects information about the PLC and creates a behavior profile. A generic web server then references that profile in order to respond properly to requests. To measure the ability of the process, the resulting emulator is evaluated based on response accuracy and timing accuracy. In addition, the configuration time of the process itself is measured. For the accuracy measurements a workload of 1000 GET requests are sent to the index.html page of the PLC, and then to the emulator. These requests are sent at two rates: Slow and PLC Break. The emulator responses are then compared to those of the PLC baseline.
Results show that the process completes in 9.8 seconds, on average. The resulting emulator responds with 97.79% accuracy across all trials. It responds 1.3 times faster than the real PLC at the Slow response rate, and 1.4 times faster at the PLC Break rate. Results indicate that the automated process is able to create an emulator with an accuracy that is comparable to a manually configured emulator. This supports the hypothesis that creating an automated process for configuring a PLC emulator with a high level of accuracy is feasible.

15. SUBJECT TERMS

SCADA, honeypot, programmable logic controller, industrial control systems, automation, emulator

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins (ENG)
					19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	82	(937) 255-3636 x7979 Barry.Mullins@afit.edu